ランゲージリファレンス

はじめに
プログラミング言語の構成要素
デバッグ
4D環境
BLOB
HTTPクライアント
JSON
PHP
SQL
Styled Text
SVG
Webエリア
Webサーバ
Webサービス (クライアント)
Webサービス (サーバ)
XML
XML DOM
XML SAX
インポート＆エクスポート
ウィンドウ
オブジェクト(フォーム)
オブジェクト(ランゲージ)
クイックレポート
クエリ
グラフ
コンパイラ
サブレコード
システムドキュメント
システム環境
ストラクチャアクセス
スペルチェッカー
セット
セレクション
ツール
データベースメソッド
データ入力
テーブル
デザインオブジェクトアクセス
ドラッグ＆ドロップ
トランザクション
トリガ
バックアップ
ピクチャ
ブール
フォーミュラ
フォーム
フォームイベント
プロセス
プロセス (コミュニケーション)
プロセス (ユーザインタフェース)
ペーストボード
メッセージ
メニュー
ユーザ＆グループ
ユーザーインターフェース
ユーザフォーム
ランゲージ
リストボックス
リソース
リレーション
レコード
レコードロック
入力制御
割込
印刷
命名セレクション
変数
文字列
日付と時間
暗号化プロトコル
演算子
算術関数
統計関数
通信
配列
階層リスト
定数テーマリスト
エラーコード
文字コード
新着
コマンドリスト (文字順)

はじめに

- 概要
- イントロダクション
- 4D アプリケーションのビルド

概要

4Dは独自のプログラミング言語を持っています。約1000にもおよぶこのビルトインの言語により、4Dはデスクトップ上のデータベースアプリケーションを作成する、パワフルな開発ツールとなっています。4D言語を使用して、単純な計算を行ったり、複雑なカスタムインターフェースを作成したり、多くの様々なタスクを行うことができます。例えば以下のようなことが可能です:

- プログラムで、クエリや並び替えなどのレコード管理エディタにアクセスする、
- データベースの情報から、複雑なレポートやラベルを作成、印刷する、
- 他のデバイスと通信する、
- ドキュメントを管理する
- 4Dデータベースと他のアプリケーション間で、データの書き出しや読み込みを行う、
- 4Dのプログラミング言語に、他の言語で書かれたプロシージャを組み込む。

4Dプログラミング言語は柔軟性とパワーを備え、あらゆるレベルのユーザや開発者がさまざまな情報管理業務を達成するための理想的なツールです。初心者ユーザでも計算処理を素早く行うことができます。またプログラミング経験がなくても、ある程度コンピュータの知識を持っているユーザであれば、自分のデータベースをカスタマイズできます。熟練した開発者であれば、4Dの強力なプログラミング言語を使用して、ファイル転送や通信などの高度な機能をデータベースに組み込むことができます。他の言語でプログラミング経験がある開発者は、独自のコマンドを4Dに追加することができます。

4Dプログラミング言語は、4Dモジュールをアプリケーションに組み込むことで追加できます。それぞれのモジュールは、専用のコマンドを持っています。

マニュアルについて

ここで紹介するマニュアルは、4Dと4D Server両方の機能を紹介しています。4D Server Referenceでは、4D Serverに特化した機能を紹介します。

- Language Referenceは4D言語利用ガイドです。4Dのコマンドや関数を組み込んで、データベースをカスタマイズする方法を学ぶためにこのマニュアルを使用します。
- Design Referenceは、デザインモードで利用可能なエディタやツールの詳細な説明を提供します。
- Self-trainingマニュアルで、4Dデータベースを作成・利用するための、例題に基づいたレッスンを行うことができます。この例題を通して、4Dや4D Serverのコンセプトや機能に慣れていただくことができます。
- 4D Server Referenceは、4D Serverによるマルチユーザデータベースの管理について説明します。

このマニュアルについて

このマニュアルは4Dランゲージについて説明しています。このマニュアルの読者はテーブル、フィールド、フォームなどの用語についてはすでにご存じであると仮定しています。このマニュアルを読む前に:

- Self-trainingマニュアルを使用して、データベースの例題を扱う
- 必要に応じてDesign Referenceを参照しながら、実際にデータベースを作成する
- 4D Webサイト (http://www.4d.com/) のデモや例題データベースを参考に、知識を深めることをお勧めします。

表記方法について

このマニュアルでは、いくつかの表記規則が使用されています:

- 4Dメソッドエディタにならい、CREATE RECORDのようにコマンドはすべて大文字で記述されます。値を返す関数は、Change stringのように大文字で始まり、小文字が続きます。
- コマンドシンタックス中、{ } 文字 (中括弧) はその引数が省略可能であることを示します。例えばSET DEFAULT CENTURY (century{; pivotYear}) はpivotYear 引数が省略できることを意味します。
- コマンドシンタックス中、| 文字は引数を候補から選択可能であることを意味します。例えば Table (tableNum | aPtr) は、関数が引数としてテーブル番号またはテーブルへのポインタを受け入れることを示します。
- このドキュメントの特定の例題では、一行のコードがスペースの都合上2行以上に分割されている場合があります。しかしながら、実際にメソッドを記述する際は、このコードを改行なしで1行に記述しなければなりません。

次は？

このマニュアルをはじめてお読みの場合は、イントロダクションに進んでください。

イントロダクション

このトピックでは、4Dプログラム言語について紹介します。以下の話題について取り上げます:

- 言語とは何か、また何ができるのか
- メソッドをどのように利用するか
- 4Dでどのようにしてアプリケーションを作成するか

この節ではこれらのトピックの一般的な事柄について説明します。詳細については他の節で説明しています。

言語とは?

4D言語は、私たちが日ごろ話している言語とさして変わりありません。4D言語はアイデアを表現したり、伝達したり、指示したりするために使用される、コミュニケーションの形です。話し言葉のように、4Dは独自のボキャブラリ、文法、シンタックスを持っています。これを使用して、開発者は4Dに、データベースやデータをどのように管理するかを伝えることができます。

4Dを効率的に使用する目的では、言語のすべてを知っている必要はありません。話すために英語のすべてを知る必要はありません。実際、少々のボキャブラリしか持っていなくても、雄弁に語ることはできるものです。4D言語も同様です。創造性を発揮するためには、言語の小さい部分を知っていればよく、他の部分はそれが必要になったときに学べばよいのです。

なぜ言語を使用するのか?

4Dを使い始めのころには、プログラミング言語の必要性は少ないように見えます。デザインモードで、4Dはプログラムを必要としない、広範囲の様々なデータ管理タスクを行うための自由度の高いツールを提供しています。データ入力やクエリ、並び替え、レポート作成などの基本的なタスクは簡単に処理できます。実際、データ検証や入力補助、グラフ作成、ラベルの生成など、多くの機能が追加で提供されています。

ではなぜ4D言語を使用するのでしょうか? 例えば以下のような利用が考えられます:

- 自動的な繰り返しタスク: このようなタスクにはデータの更新、複雑なレポートの生成、長い一連の操作を全自動で行うといったことが含まれます。
- ユーザインタフェースのコントロール: ウィンドウやメニューを管理したり、フォームやインタフェースオブジェクトをコントロールできます。
- 高度なデータ管理を行う: このようなタスクにはトランザクション処理、複雑なデータ検証、マルチユーザ管理、セットや命名セレクションの処理が含まれます。
- コンピュータのコントロール: シリアルポート通信やドキュメント管理、エラー管理を行うことができます。
- アプリケーションの作成: アプリケーションモードで動作する、カスタマイズされたデータベースを作成できます。
- 組み込みの4D Webに機能を追加する: 4Dが自動で変換するWebフォームに動的なHTMLページを追加できます。

言語を使用すれば、データベースのデザインや処理を完全にコントロールできます。4Dはパワフルな汎用のエディタを提供していますが、言語を使用すればデータベースを必要に応じてカスタマイズできるのです。

データをコントロールする

4D言語を使用すれば、パワフルでエレガントに、データをコントロールできます。4D言語は初心者にも使いやすく、経験豊かな開発者の利用に耐えるほど高度です。言語の利用により、組み込みのデータベースの機能から、完全にカスタマイズされたデータベースにスムースに移行できます。

4D言語のコマンドを使用して標準のレコード管理エディタにアクセスできます。例えばQUERY コマンドを使用すれば (デザインモードでアクセスできる) クエリエディタが表示されます。この言語コマンドを利用することは、**レコード**メニューの**クエリ**コマンドを選択するのと同じくらい簡単ですが、QUERYコマンドはさらに利用価値があります。QUERY コマンドを使用して、指定したデータを検索できるのです。例えばQUERY ([People];[People]Last Name="Smith") はデータベースから"Smith"という名前の人物をすべて検索します。

4D言語はとてもパワフルです。ひとつのコマンドはしばしば従来のプログラム言語で書かれた数百行あるいは数千行にも及ぶコードに相当します。このパワーを持ちながら、コマンドにはシンプルな英語の名称がつけられています。例えばクエリを行うにはQUERYコマンドを、レコードを追加するにはADD RECORDコマンドを使用します。

言語は、ほとんどどんなタスクでも簡単に実行できるようデザインされています。レコードの追加、並び替え、データの検索などの操作がシンプルなコマンドで提供されています。さらにコマンドを使用してシリアルポートのコントロール、ディスク上のドキュメントの読み込み、高度なトランザクション処理等を行うこともできるのです。

4D言語はもっと高度なタスクを相対的に簡単に行います。このようなタスクを言語なしで行うことはほとんど想像できません。
言語のパワフルなコマンドを使用したとしても、タスクによっては複雑で難しいものとなることがあります。ツール自身はタスクを処理しません。タスク自身がチャレンジすべきものであり、ツールはその処理を簡単にするだけです。例えば、ワープロを使用すれば素早く簡単に文章を書くことができますが、ワープロ自体が文章を書くわけではありません。4D 言語の使用はデータの管理処理を簡単にし、複雑なタスクの処理に自信を持ってアプローチできるようにします。

4D言語は従来のコンピュータ言語ですか?

従来のコンピュータ言語に馴れ親しんでいる方は、本節を参照してください。それ以外の方は、この章を読み飛ばしても構いません。

4D言語は従来のコンピュータ言語とは異なります。この言語は、今日のコンピュータで使用できる最も先進的で柔軟性のある言語の1つです。4Dのプログラミング言語は、他の方法ではなく、あなたが行った通りに動作するよう設計されています。

従来の言語を使用して開発を行う場合、まず広範な計画を立てる必要があります。実際計画の立案は開発の重要な工程の1つとなります。4Dはデータベースのあらゆる部分でプログラミング言語をいつでも使用することができます。例えば、初めにフォームにメソッドを追加し、後ほどさらにいくつかのメソッドを追加することができます。データがより高度になったら、メニューから制御するプロジェクトメソッドを追加することもできます。必要最低限の言語を使用できます。他の多くのデータベースのような"すべてか、あるいは全くないか"ではありません。

従来の言語では、オブジェクトを正式に定義しなければなりません。しかし、4Dではボタンなどのオブジェクトを作成し、それを使用するだけで構いません。4Dは自動的にオブジェクトを管理します。例えば、ボタンを使用するためには、ボタンをフォーム上に作成し、名前を指定します。ユーザがボタンをクリックした時点で、メソッドに通知します。

従来の言語では、コマンドの使用を固定化したり限定する等して融通性に欠ける点が多いのに対して、4Dのプログラミング言語は優れたユーザインタフェースを実現しています。

メソッドはプログラミング言語の入り口

メソッドは、4Dにタスクを実行させるための一連の命令文です。メソッド内の各行を"ステートメント"と呼びます。各ステートメントは、プログラミング言語の部品で構成されます。

ここでは、既にSelf-Trainingを通読し、メソッドを作成し使用した経験があることを前提に説明を行います。

4Dには次の5種類のメソッドがあります:

- **オブジェクトメソッド**: フォームオブジェクトを制御するために使用する短いメソッド。
- **フォームメソッド**: フォームの表示、印刷を管理するメソッド。
- **トリガ**: データベースの規則を強制するためのメソッド。
- **プロジェクトメソッド**: データベース全体で使用できるメソッド。例えばメニューに関連付けるメソッドなど
- **データベースメソッド**: データベースのオープンやクローズのとき、またはWebブラウザがインターネットおよびイントラネット上でWebサーバとして公開されているデータベースに接続するときに、初期化や特別な動作を実行するメソッド

次節では、各メソッドの紹介とデータベースを自動化する方法について説明します。

オブジェクトメソッドについて

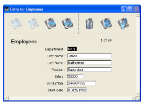
動作を行うことが可能なフォーム・オブジェクト(アクティブオブジェクト)は、関連するメソッドを持つことができます。オブジェクトメソッドは、データ入力時や印刷時にアクティブオブジェクトの監視や管理を行います。オブジェクトをコピーして貼り付けると、オブジェクトメソッドがそのアクティブオブジェクトとともにコピーされます。これにより、スクリプトが付けられたオブジェクトの再利用可能なライブラリを作成することができます。オブジェクトメソッドは必要な時にコントロールを得ます。

オブジェクトメソッドは、データベースへの入り口であるユーザインタフェースを管理するための主要ツールです。ユーザインタフェースは、コンピュータがユーザと通信するための手順とルールから成り立っています。その最終目的は、データベースのユーザインタフェースをできるだけ簡単で使いやすくすることです。ユーザインタフェースは、コンピュータとのやり取りを快適にし、ユーザがそれを楽しめるように、または気にならないようにする必要があります。

フォームには、以下の2つの基本的なタイプのアクティブオブジェクトがあります:

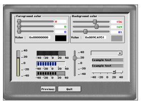
- フィールド等のデータの入力、表示、格納のために使用されるもの
- 入力エリア、ボタン、スクロールエリア、階層リスト、メータ等コントロールに使用されるもの

4Dでは、以下のようなクラシックフォームを作成できます:

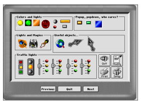


Department : MAN
First Name : James
Last Name : Rutherford
Position : Supervisor
Salary : 55000
SS Number : 244984392
Start date : 01/05/1992

また、以下のような複数のグラフィックコントロールを使用してフォームを作成できます:



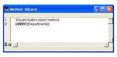
想像力が許す限りのグラフィックを使用したフォームを作成することもできます:



作成するスタイルがどのような形式であろうと、アクティブオブジェクトはすべてデータ入力エリアの範囲チェックや入力フィルタ、コントロール、メニュー、ボタン用の自動動作等の内蔵の支援機能を持っています。オブジェクトメソッドを追加する前にこれらの支援機能を使用出来ます。内蔵の支援機能はメソッドと同様に、アクティブオブジェクトに関連付けられ、それがアクティブなのはアクティブオブジェクトが使用されるときだけです。
通常、内蔵の支援機能とオブジェクトメソッドを組み合わせて使用して、ユーザインタフェースを制御します。

データ入力に使用するオブジェクトメソッドは、フィールドまたは変数に対して特定の処理を実行します。このオブジェクトメソッドは、データの属性チェック、フォーマット編集、計算処理等を行うことが出来ます。別テーブルのデータを情報を取得することもできます。もちろん、これらの処理は、4Dに内蔵されたデータ入力支援により実行することも出来ます。しかし、複雑な処理が必要な場合には、オブジェクトメソッドを使用します。内蔵されたデータ入力支援に関する詳細は4D Design Referenceを参照してください。

オブジェクトメソッドは、制御に使用するボタン等のオブジェクトにも作成することができます。これらのオブジェクトは、データベースを使用する上でとても重要です。これには、レコード移動や別フォームへの移動、データの追加、削除等を実行するボタンがあります。また、これらのオブジェクトは、データベースの使用を簡単にし、それを習得するために必要な時間を減少させます。ボタンにも、データ入力時と同様に内蔵の支援機能が使用可能です。オブジェクトメソッドを使用すれば、4D内に組み込まれていない動作を追加することができます。例えば、以下の図は、クリックされた時点でクエリエディタを表示するボタンのオブジェクトメソッドを示しています。



```
`bQuery button object method
QUERY([Departments])
```

オブジェクトメソッドに精通してくると、オブジェクトメソッドを持つオブジェクトのライブラリを作成すると便利なことに気付きます。これらのオブジェクトやオブジェクトメソッドをフォーム、テーブル、データベース間でコピーしたり、貼り付けることができます。また、必要な場合に使用できるように、クリップボード(Windows)またはスクラップブック(Macintosh)にそれらを保存することもできます。

フォームメソッドを使ってフォームをコントロールする

オブジェクトメソッドがフォームオブジェクトに付随するのと同様に、フォームメソッドはフォームに付随します。1つのフォームには、1つのフォームメソッドを持つことができます。フォームはデータの入力、表示および印刷をする手段です。フォームは様々な方法でデータを見せてくれます。フォームを使用することで、魅力的で使いやすいデータ入力画面や印刷帳票を作成することができます。フォームメソッドはデータの入力画面や印刷するための個々のフォームを監視および管理するのに使用します。

フォームメソッドは、オブジェクトメソッドよりも上位のレベルでフォームを管理します。オブジェクトメソッドは属しているオブジェクトが使用された場合にのみ実行されるのにたいし、フォームメソッドはフォームのあらゆる動作で実行されます。そのためフォームメソッドは、フォーム上の異なるオブジェクト間の相互関係やフォーム全体を制御するために使用します。

フォームは多くの方法で使用されるため、フォームが使われたときに何が起こったのかを把握することが必要です。この目的のために**フォームイベント**を使用します。フォームイベントは、フォームに現在何が起こったのかを伝えます。フォーム上のオブジェクトメソッドと同様、個々のタイプのイベント(クリック、ダブルクリック、キーストローク等)毎にフォームメソッド実行の可否を設定することができます。

フォーム、オブジェクト、イベント、そしてメソッドについての詳細は、を参照してください。

トリガを使ってデータベースのルールを強制する

トリガはテーブルに設定されるため、テーブルメソッドとも呼ばれます。トリガは、テーブルのレコードを操作(追加、削除、修正)するたび、4Dデータベースエンジンによって自動的に起動されます。トリガはデータベースのレコードに対する「違法な」操作を防止することができます。例えば、請求書発行システムでは、請求書発行先の顧客を指定しない請求書を追加できないようにすることができます。トリガはテーブル上の操作を制限する非常に強力なツールであると同時に、偶発的なデータの損失や不正な変更を防止します。最初は簡単なトリガを作成しておいて、後で高度にしていくこともできます。

トリガについての詳細は、を参照してください。

データベース全体で使われるプロジェクトメソッド

特定のオブジェクトやフォーム、テーブルに付随するオブジェクトメソッド、フォームメソッド、トリガと異なり、プロジェクトメソッドはデータベースのどこからでも使用することができます。プロジェクトメソッドは繰り返し使用したり、別のメソッドの中から呼び出して使用することもできます。同じ処理を繰り返し実行する必要がある場合でも、それぞれに対して同一のメソッドを書く必要はありません。プロジェクトメソッドは、別のプロジェクトメソッドやオブジェクトメソッド、フォームメソッドから必要な場所で随時呼び出されます。呼び出されたプロジェクトメソッドは、呼び出した場所にメソッド全体を書き込んだ時と同じように動作します。別のメソッドから呼び出されるプロジェクトメソッドのことを"サブルーチン"と呼びます。

プロジェクトメソッドを使用するもう1つの方法として、メニューへの割り当てがあります。メニューに割り当てられたメソッドは、メニューを選択した時に実行されます。メニューは、プロジェクトメソッドを呼び出すものと考えることができます。

データベースを管理するデータベースメソッド

フォームにイベントが発生したときに、オブジェクトメソッドやフォームメソッドが起動されるのと同様に、データベースへのイベント発生により起動するメソッドがあります。これが**データベースメソッド**です。例えば、データベースをオープンするたびに、データベース全体で使用する変数を初期化したい場合があります。これには、データベースをオープンしたときに4Dが自動的に実行するOn Startupデータベースメソッドを使用します。

データベースメソッドについての詳細はCLEAR SEMAPHOREを参照してください。

データベースを開発する

開発は、言語や内蔵ツールを使用してデータベースをカスタマイズするということです。

データベースを作成すると、プログラミング言語の第一歩を踏み出したことになります。データベースのテーブル、フィールド、フォーム、オブジェクト、メニューの各部分は、プログラミング言語と密接に関連しています。4D言語は、データベースのこれらの部分について対応します。

プログラミング言語を初めて使用する場面は、おそらくデータ入力を制御するために、フォームオブジェクトにオブジェクトメソッドを作成する場合でしょう。そして、フォームの表示を制御するために、フォームメソッドを作成します。データベースが複雑になってくると、データベースを完全にカスタマイズするために、プロジェクトメソッドを割り当てたメニューバーを追加できます。

4Dによる開発は非常に柔軟性があります。データベースの作成に、決まった方法があるわけではありません。自分自身に合った方法で作成することができます。もちろん、以下のようないくつかの一般的なパターンはあります。

- 実装: 設計した内容をデザインモードで実装する。
- テスト: アプリケーションテストメニューコマンドを使用してアプリケーションモードに移動し、設計した内容やカスタマイズした要素をテストします。
- 使用: データベースのカスタマイズが完了したら、データベースを直接アプリケーションモードで実行するよう設定します。
- 修正: エラーを発見したら、デザインモードに移動してそれを修正します。

4Dには、必要になるまで隠されている、データベース作成のための特別な開発支援ツールが内蔵されています。プログラミングに熟知するにつれて、これらのツールを活用することで開発プロセスを容易にしてくれることがわかってきます。例えば、メソッドエディタはタイプミスを見つけたり、プログラムソースを適切に整形します。またインタプリタ(プログラムの実行エンジン)はシンタックス中のエラーを見つけ出し、どこが誤りかを示します。デバッガは、メソッド中のエラーを見つけるために、実行中のメソッド監視、表示することができます。

アプリケーションのビルド

これでデータ入力、検索、ソート、レポート等データベースの一般的な使用方法について理解されたと思います。これらの処理は、4D内に組み込まれている標準のメニューやエディタを使用してデザインモードで実行してきました。

データベースを使用するにつれて、繰り返して実行する一連の処理があることが明らかになります。例えば個人連絡先のデータベースにおいて、データを変更するたびに、取引先を検索、名字でソート、特定のレポートを印刷すると仮定します。このようなタスクは難しくないように見えますが、処理を20回以上も実行するとなると、時間を消費します。また数週間データベースを使用しなかった場合には、レポートを作成する手順を克明に憶えているとも限りません。しかし、メソッドを作成すれば、1つのメニューを選択するだけで、一連の処理を自動的に実行してくれるためレポートを作成するための手順を忘れていても大丈夫です。

アプリケーションはカスタムメニューを持ち、データベース利用者の要求する処理を行います。アプリケーションは、データベースの全ての要素(ストラクチャ、フォーム、オブジェクト、メソッド、メニュー、パスワード等)から構成されます。

データベースをコンパイルし、スタンドアロンのWindowsとMacintoshアプリケーションを作成できます。データベースをコンパイルすることは言語の実行速度を向上させ、データベースを保護し、完全に独立したアプリケーションを作成することを可能にします。4Dに内蔵されたコンパイラは整合性のためメソッド内で変数のシンタックスと変数の型をチェックします。

アプリケーションは、人の名前を入力してレポートを印刷するだけの単純なものから、請求・在庫管理システムといった複雑なものまであります。アプリケーションの使用に制限はありません。一般にデータベースは、デザインモードでの使用から、カスタムメニューやフォームで完全にコントロールするように拡張していきます。

次は?

- アプリケーションの開発は、開発者の思うままに複雑にも簡単にもすることもできます。簡単な4Dアプリケーションの構築については4Dアプリケーションのビルドを参照してください。
- 4Dを始めたばかりの方は、4Dの言語の基本について学ぶために4Dプログラミング言語へお進みください。

4Dアプリケーションのビルド

アプリケーションは、特定の要求を満たすために設計されたデータベースです。アプリケーションは、操作を容易にするようにデザインされたユーザインタフェースを持っています。あるアプリケーションが実行する作業は、その目的内に限定されます。4Dを使用すると、アプリケーション作成は従来のプログラミングよりも速やかに、かつ容易に行うことができます。4Dは、次に示すようなさまざまなアプリケーション作成に利用可能です:

- 請求書システム
- 在庫管理システム
- 会計システム
- 給与システム
- 人事システム
- 顧客管理システム
- インターネットやイントラネット経由による共有データベース

これらすべてのシステムを一つのアプリケーションに納めることもできます。このようなアプリケーションは、データベースの代表的な利用例です。さらに、4Dのツールを使用すると、次のような画期的なアプリケーションを作成することができます:

- 文書管理システム
- 画像管理システム
- カタログ発行アプリケーション
- シリアルデバイス制御／監視システム
- 電子メールシステム（E-メール）
- マルチユーザスケジューリングシステム
- メニューリスト、ビデオコレクション、音楽コレクション等の一覧

通常、アプリケーションは、デザインモードで使用するデータベースから始めることができます。カスタマイズするにつれて、データベースがアプリケーションへと"進化"していきます。アプリケーションの異なる点は、データベース管理に必要となるシステムがユーザの目に触れないというところです。データベース管理は自動化され、ユーザはメニューを利用して特定の作業を実行します。

4Dデータベースをデザインモードで使用する場合、結果を実現するために必要となる手順を知っておかなければなりません。アプリケーションのアプリケーションモードを使用するには、デザインモードでの自動化されている以下のようなすべての事柄を、あなたが管理する必要があります。

- テーブル操作：ユーザはテーブルリストウィンドウや最後に使用したテーブル、ナビゲーションボタンを使用できません。メニューコマンドとメソッドを使用して、テーブル操作を制御します。

- メニュー：アプリケーションモードには、終了メニューコマンドが納められたデフォルトのファイルメニューの他、編集、モード、そしてヘルプメニュー、(Mac OSではアプリケーションメニューも) が存在します。アプリケーションでさらにメニューが必要となる場合、メニューを作成し、4Dメソッドや標準アクションを用いてこれらの管理をしなくてはなりません。

- エディタ：アプリケーションモードでは、クエリや並び替え等のエディタを自動的に利用することはできません。これらのエディタを使用したい場合、4Dメソッドを用いて呼び出す必要があります。

以下の節では例題を用いて、プログラミング言語によってデータベースの使用を自動化する方法を紹介します。

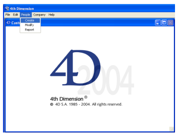
アプリケーションモード: 例題

カスタムメニューは、アプリケーションにおける主要なインタフェースです。このメニューにより、ユーザはより簡単にデータベースの習得や利用を行えるようになります。カスタムメニューの作成はとても簡単です。メニューエディタを使用して、各メニューコマンド（メニュー項目とも呼ばれる）にメソッドや自動アクションを関連付けるだけです。

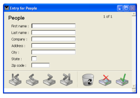
「ユーザの視点」の節では、ユーザがメニューコマンドを選択した後の事象について説明します。次の節「メニューの舞台裏」では、この動作を実現するための設計について説明します。例題はシンプルですが、カスタムメニューを用いることにより、データベースの使用や習得がいかに楽になるかが理解できます。デザインモードの"一般的な"ツールやメニューコマンドではなく、ユーザはそれぞれのニーズに合った事柄だけに注目することができます。

ユーザの視点

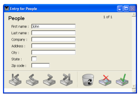
ユーザは**People**メニューから**Create**というメニュー項目を選択し、新しいPeopleをデータベースに追加します。


4th Dimension
File Edit People Company Help
Create
Modify
Report
4D Custo
4th Dimension ®
© 4D S.A. 1985 - 2004. All rights reserved.

Peopleテーブルの入力フォームが表示されます。

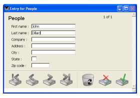

Entry for People
People
1 of 1
First name :
Last name :
Company :
Address :
City :
State :
Zip code :

ユーザはPeopleのFirst Nameを入力し、タブで次のフィールドへ移動します。


Entry for People
People
1 of 1

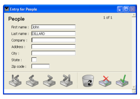
First name : John

ユーザはPeopleのLast Nameを入力します。


Entry for People
People
1 of 1

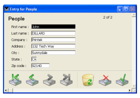
First name : John
Last name : Dillard

ユーザはタブで次のフィールドへ移動します。: Last Nameが大文字に変換されます。


Entry for People
People
1 of 1

First name : John
Last name : DILLARD

ユーザはレコードの入力を終了し、登録ボタンをクリックします（通常は、ボタンバー上の最後のボタン）。

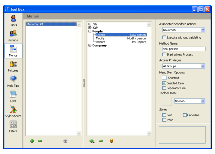

Entry for People
People
2 of 2

First name : John
Last name : DILLARD
Company : Printek
Address : 132 Tech Way
City : Sunnydale
State : CA
Zip code : 92140

空のレコードが表示されるので、ユーザはキャンセルボタン("X"印が付いたボタン)をクリックして"データ入力ループ"を終了します。再びメニューバーが表示されます。

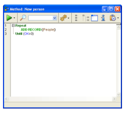
舞台裏

メニューバーは、デザインモードのメニューバーエディタを使用して作成します。


Tool Box
Menus
Users
Groups
Menus
Pictures
Help Tips
Lists
Style Sheets
Filters
Menu Bar #1
File
Edit
People
Create
Modify
Report
Company
New person
Modify person
My Report

Associated Standard Action: No Action
☐ Execute without validating
Method Name: New person
☐ Start a New Process
Access Privileges: All Groups
Menu Item Options:
☐ Shortcut
☑ Enabled Item
☐ Separator Line
Toolbar Icon: No icon
Style:
☐ Bold ☐ Underline
☐ Italic

Createメニュー項目には、New personという名前のプロジェクトメソッドが関連付けられています。このメソッドは、デザインモードのメソッドエディタを使用して作成されています。


Method: New person

```
Repeat
   ADD RECORD([People])
Until (OK=0)
```

ユーザがこのメニュー項目を選択すると、メソッドNew Personが実行されます。

```
REPEAT
   ADD RECORD([People])
Until(OK=0)
```

Repeat...Untilループ内のADD RECORDコマンドは、デザインモードにおける**新規レコード**メニュー項目と同様の動作を行います。このループは入力フォームを表示し、ユーザは新規レコードを追加することができます。ユーザがレコードを保存すると、新たに空のレコードが表示されます。このADD RECORDループは、ユーザが「キャンセル」ボタンをクリックするまで実行を続けます。

レコードに入力すると、以下の作業が行われます:

- First Name フィールドにはメソッドがなく、何も実行されません。
- Last Name フィールドにはメソッドがあります。このオブジェクトメソッドはデザインモードのフォームエディタとメソッドエディタで作成されます。メソッド内では以下のコードが実行されます:

```
Last Name:=Uppercase([People]Last Name)
```

このコードはLast Name フィールドの内容を大文字に変換します。

レコードが入力され、ユーザが次のレコードでキャンセルボタンをクリックすると、OKシステム変数が0に設定されてADD RECORDループが終了します。

次に実行するステートメントが存在しないため、New Personメソッドは実行を終了し、制御がメニューバーに戻ります。

自動化された処理と、デザインモードで実行される動作との比較

あるタスクをデザインモードで実行する方法と、プログラミング言語で実行する方法とを比較してみましょう。実行する処理は共通のものです。

- レコードグループの検索
- 並び替え
- レポートの印刷

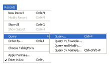
次の節の"デザインモードにおけるデータベースの使用"では、デザインモードで実行される処理を示します。

その次の節、"アプリケーションモードにおける組み込みエディタの使用"では、同じ処理がアプリケーションで実行される様子を示します。

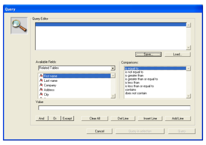
これらのメソッドはともに同じ処理を実行しますが、2番目の節の手順はプログラミング言語を使用して自動化されている点に注目してください。

デザインモードにおけるデータベースの使用

ユーザは**レコード**メニューから**クエリ>クエリ...** を選択します。

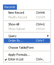

Records
New Record Ctrl+N
Modify Record Ctrl+M
Show All Ctrl+G
Show Subset Ctrl+H
Query
Order By... Ctrl+T
Choose Table/Form
Apply Formula...
✓ Enter in List Ctrl+,
Query... Ctrl+F
Query by Example...
Query and Modify...
Query by Formula... Ctrl+Shift+F

クエリエディタが表示されます。

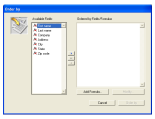

Query
Query Editor
Save...
Load...
Available Fields:
Related Tables
First name
Last name
Company
Address
City
Comparisons:
is equal to
is not equal to
is greater than
is greater than or equal to
is less than
is less than or equal to
contains
does not contain
Value
And
Or
Except
Clear All
Del Line
Insert Line
Add Line
Cancel
Query in selection
Query

ユーザは検索条件を入力し、**クエリ**ボタンをクリックします。検索が行われます。

ユーザは**レコード**メニューから**並び替え**を選択します。


Records
New Record Ctrl+N
Modify Record Ctrl+M
Show All Ctrl+G
Show Subset Ctrl+H
Query
Order By... Ctrl+T
Choose Table/Form
Apply Formula...
✓ Enter in List Ctrl+,

並び替えエディタが表示されます。


Order by
Available Fields
First name
Last name
Company
Address
City
State
Zip code
Ordered by Fields/Formulas
Add Formula...
Modify...
Cancel
Order by

ユーザが並び替え条件を入力して**並び替え**ボタンをクリックすると、並び替えが実行されます。

この後、レコードを印刷するには、さらに次の手順が必要となります:

- ユーザが**ファイル**メニューから**プリント**...を選択します。
- いずれのフォームを印刷するかをユーザが認識する必要があるため、**プリント**ダイアログボックスが表示されます。
- 印刷用ダイアログボックスが表示されます。ユーザは各設定を選択し、レポートが印刷されます。

アプリケーションモードでビルトインのエディタを使用する

前述の作業がアプリケーションモードではどのように行われるかを検証しましょう。

ユーザは**People**メニューから**Report**を選択します。

すでにこの時点で、ユーザにとってアプリケーションを使用するほうが簡単です。ユーザは最初の手順がクエリであることを知っている必要はありません。

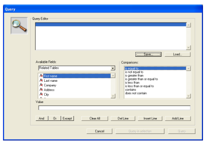
メニューコマンドにはMy Reportというメソッドが関連付けられています。このメソッドは次の通りです:

```
QUERY([People])
ORDER BY([People])
FORM SET OUTPUT([People];"Report")
PRINT SELECTION([People])
```

最初の行が実行されます:

```
QUERY([People])
```

クエリエディタが表示されます。


Query
Query Editor
Save...
Load...
Available Fields:
Related Tables
First name
Last name
Company
Address
City
Comparisons:
is equal to
is not equal to
is greater than
is greater than or equal to
is less than
is less than or equal to
contains
does not contain
Value
And
Or
Except
Clear All
Del Line
Insert Line
Add Line
Cancel
Query in selection
Query

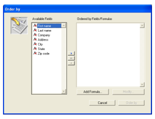
ユーザが検索条件を入力し**クエリ**ボタンをクリックすると、検索が実行されます。

次に、メソッドMy Reportの2行目が実行されます:

```
ORDER BY([People])
```

ユーザは次の手順がレコードの並び替えであることを認識する必要がない点に注目してください。

並び替えエディタが表示されます。


Order by
Available Fields
First name
Last name
Company
Address
City
State
Zip code
Ordered by Fields/Formulas
Add Formula...
Modify...
Cancel
Order by

ユーザが並び替え条件を入力し**並び替え**ボタンをクリックすると、並び替えが実行されます。

次に、メソッドMy Reportの3行目が実行されます:

```
FORM SET OUTPUT([People];"Report")
```

ここでもユーザは、次に実行されることを知っている必要がありません。このメソッドがすべて処理します。

最後に、メソッドMy Reportの最終行が実行されます:

```
PRINT SELECTION([People])
```

印刷ダイアログボックスが表示されます。ユーザが各設定を選択すると、レポートが印刷されます。

アプリケーションをもっと自動化する

前述の例題で使用したものと同じコマンドを使用し、データベースの自動化をさらに進めることができます。

新しくなったMy Reportメソッドを見てみましょう。

ユーザは、**People**メニューから**Report**を選択します。メニューコマンドにはMy Report2というメソッドが関連付けられています。このメソッドは次の通りです:

```
QUERY([People];[People]Company="Acme")
ORDER BY([People];[People]Last Name;>;[People]First Name;>)
FORM SET OUTPUT([People];"Report")
PRINT SELECTION([People];*)
```

最初の行が実行されます:

```
QUERY([People];[People]Company="Acme")
```

クエリエディタは表示されません。その代わりに、QUERYコマンドにより検索条件が指定され、実行されます。ユーザは何も行う必要はありません。

メソッドMy Report2の2行目が実行されます:

```
ORDER BY([People];[People]Last Name;>;[People]First Name;>)
```

並べ替えエディタは表示されず、並び替えが即座に実行されます。ここでもユーザによる処理は必要ありません。

メソッドMy Report2の最終行が実行されます:

```
FORM SET OUTPUT([People];"Report")
PRINT SELECTION([People];*)
```

印刷ダイアログボックスは表示されません。PRINT SELECTIONコマンドは、任意の引数であるアスタリスク(*)を受け入れ、これによりコマンドはレポートフォームの作成時に指定された印刷設定を使用し、レポートが印刷されます。

このように自動化をさらに進めることにより、ユーザは3つのダイアログボックスでオプションを入力する必要がなくなりました。これらの利点は次の通りです:

- クエリが自動的に実行される:ユーザが指定すると、クエリの作成時に誤った条件を選択する可能性があります。
- 並び替えが自動的に実行される:ユーザが指定すると、並び替えの定義時に誤った条件を選択する可能性があります。
- 印刷が自動的に実行される:ユーザが指定すると、誤ったフォームを選択して印刷する可能性があります

4Dアプリケーション開発のためのヘルプ

4Dアプリケーションの開発を進めていくと、最初は気付かなかった数多くの機能を発見することでしょう。また、他のツールやプラグインを4D開発環境に追加し、標準の4Dの機能を高めることができます。

4Dプラグイン

4Dはアプリケーションの機能を向上させる各種プラグインを提供しています。

- **4D Write**: ワードプロセッサ
- **4D View**: スプレッドシートおよびリストエディタ
- **4D Internet Commands**: インターネット上の通信用ユーティリティ
- **4D Pack**: 4Dに30以上のコマンドを追加します。
- **4D ODBC Pro**: ODBCを使用した接続
- **4D for OCI**: ORACLE Callインタフェースを用いた接続

詳細に関しては、4D社または4Dパートナーまでお問い合わせください。Webサイトでもこれらの情報を提供しています。:
http://www.4d.com/jp/

4Dコミュニティおよびサードパーティツール

世界各国において、ユーザグループや電子フォーラム、4Dパートナーから構成されている4Dコミュニティが意欲的に活躍しています。4Dパートナーからは**サードパーティツール**が提供されています。次のアドレスで、4Dのユーザフォーラムに登録することができます:
http://forums.4D.fr/

4Dコミュニティからは、ヒントやTips、ソリューション、情報、その他のツールが提供され、開発にかかる時間やエネルギーを節約して生産性を高めることが可能です。

プログラミング言語の構成要素

- 4Dプログラミング言語
- データタイプ
- 定数
- 変数
- システム変数
- ポインタ
- 識別子
- 制御フロー
- If...Else...End if
- Case of...Else...End case
- While...End while
- Repeat...Until
- For...End for
- メソッド
- プロジェクトメソッド

4Dプログラミング言語

4Dのプログラミング言語は、以下のような要素から構成されています。これらの要素は、処理を実行、またはデータを管理する際の手助けをします。

- データタイプ：データベース内のデータの分類。詳細はこの章の後述およびの章を参照してください。
- 変数：メモリ内でデータを一時的に記憶する領域。詳細はの章を参照してください。
- 演算子：2つの値を計算するために使われる記号。詳細はこの章の後述およびの章を参照してください。
- 式：値を求めるための構成要素の組み合わせ。詳細はこの章の後述を参照してください。
- コマンド：処理を実行するために4Dに組み込まれている命令文。すべての4Dコマンド、例えばADD RECORD コマンドなどはテーマ別にこのマニュアルに記載されています。また必要な場合は、序章に続きテーマの記載があります。4Dプラグインを使い、4Dの開発環境に新規のコマンドを追加することもできます。例えば、4Dシステムに4D Writeを追加すると、4D Writeのコマンドを利用してワードプロセスのドキュメントを作成および操作を行うことが可能になります。
- メソッド：このマニュアルに記載されたプログラミング言語による命令文。詳細はの節および関連項目を参照してください。

この章では、データタイプ、演算子、式について説明します。その他の内容については、上記の章を参照してください。

追記：

- 言語を構成する要素、例えば変数には識別子がつけられています。識別子についての詳細およびオブジェクトに名前をつける場合の規則についてはの章を参照してください。
- 配列変数についての詳細はの章を参照してください。
- BLOB変数についての詳細はの章を参照してください。
- データベースをコンパイルする場合は、の章および4D Design Referenceマニュアルを参照してください。

データタイプ

4Dのデータベースで使用するデータには、いくつかの種別があります。これらのデータ種別を"データタイプ"と呼びます。
4Dには、以下のような7つの基本的なデータタイプ（文字、数字、日付、時間、ブール、ピクチャーBLOB）とポインタがあります。

- **文字列**："こんにちは"等の一連の文字。文字フィールドとテキストフィールドは、ともに文字列タイプです。
- **数値**：2や1,000.67等の数値。整数フィールド、倍長整数フィールド、実数フィールドはすべて数値タイプです。
- **日付**：97/07/20等の日付。日付フィールドは、日付タイプです。
- **時間**：1:00:00や4:35:30 P.M.等の時:分:秒。時間フィールドは、時間タイプです。
- **ブール**："True（真）"または"False（偽）"のどちらかの値を返すもの。ブールフィールドは、ブールタイプです。
- **ピクチャ**：PICTタイプで作成されたピクチャ。ピクチャフィールドは、ピクチャタイプです。
- **BLOB**（Binary Large OBject）：サイズが2GB以内のバイト群。BLOBフィールドは、BLOBタイプです。
- **ポインタ**：上級プログラミングで使用する特殊タイプのデータ。対応するフィールドはありません。

注：これらのデータタイプに格納されるデータは、4D Design Referenceの中で説明されている4Dフィールドの中に格納されるデータと一致します。

データタイプの中で文字列タイプと数値タイプは、複数の類似するフィールドタイプに対応する点に注意してください。プログラミング言語でこれらのタイプのフィールドを操作する場合、プログラミング言語が自動的にサポートするデータタイプにデータを変換します。例えば、整数フィールドを使用すると、自動的に数値として扱います。つまり、プログラミング言語として使用する場合は、類似するフィールドタイプと厳密に区別する必要はありません。

しかし、プログラミング言語を使用する場合は、異なるデータタイプと混同しないようにすることは重要です。"ABC"を日付フィールドに格納しても意味がないように、日付として使用する変数に"ABC"を格納することも意味がありません。4Dは、実行したことをできるだけ有効にしようとします。例えば、日付に数値を加算した場合は、日付に日数を加算したいものと認識します。しかし、日付に文字列を加算した場合には、4Dはその操作に意味を持たないことをユーザに警告します。

あるタイプとしてデータを格納したものを、別のタイプとして使用する場合があります。
4Dのプログラミング言語には、あるタイプから別のタイプへ変換するためのコマンドが用意されています。

例えば、数値で始まり、"abc"等の文字で終了する部品番号を作成するような場合、以下のように記述することができます。

```
[Products]Part Number:=String(Number)+"abc"
```

数値が17であれば、[Prosucts]PartNumberに"17abc"という文字列が代入されます。

データタイプについてはの節で説明しています。

演算子

プログラミング言語を使用する際に、データのみを必要とする場合は非常に稀です。
データを加工、またはそれを何らかの目的のために使用することがほとんどです。例えば、演算子を使用して2つのデータから1つの新しいデータを生成する場合です。1+2は2つの数値を加算（演算子+）して、3という結果を返します。以下に、よく知られている4つの演算子を示します。

演算子	オペレーション	例
+	加算（足し算）	1 + 2の結果は3
?	減算（引き算）	3 ? 2の結果は1
*	乗算（かけ算）	2 * 3の結果は6
/	除算（割り算）	6 / 2の結果は3

数値演算子は、使用可能な演算子のうちの1つのタイプにすぎません。4Dは、数値、テキスト、日付、ピクチャ等、異なるタイプのデータを扱うために、各データタイプに対する演算を実行するための演算子を備えています。

同じ記号でも処理を実行するデータタイプによって、異なる意味に使用される場合があります。例えば、プラス記号（+）は下記のように、各データタイプによって異なる演算を実行します。

データタイプ	オペレーション	例
数値	加算（足し算）	1 + 2 は数値を加算し、結果として3になります。
文字	連結（結合）	"みなさん" + "こんにちは"連結（結合）して"みなさんこんにちは"になります。
日付と数値	日付の加算	!2006/12/4! + 20 は、2006年12月4日に20日を加算し、2006年12月24日になります。

演算子についての詳細は演算子を参照してください。

式

式は、値を返します。実際に、プログラミング言語として使用している場合は、常に式を使用します。式は、"フォーミュラ"と呼ぶこともあります。

式は、コマンド、演算子、変数、フィールド等プログラミングのほとんどの部分で使用します。式により、ステートメント（メソッドの1文や1行）を構成します。必要に応じて、メソッドの任意の場所に式を使用することができます。

式が単独で使われることはほとんどありませんが、単独で使用できる場合がいくつかあります。

- 「フォーミュラで検索」ダイアログボックス
- 「デバッガ」で式の値をチェックする
- 「フォーミュラで更新」ダイアログボックス
- 「クイックレポート」エディタでカラムをフォーミュラとして使用する

数値の4または文字列の"こんにちは"等の定数だけで構成された式の値は、常に一定で、決して変わることはありません。
演算子を使用すると、式でさまざまなことを実行することができます。前節で、演算子を使用する式を既に紹介しました。例えば、4+2は加算演算子を使用した式で2つの数値を加算し、結果として6を返します。

データタイプによって、以下の7つのタイプの式を使用することができます。

- 文字列式
- 数値式
- 日付式
- 時間式
- ブール式
- ピクチャ式
- ポインタ式

以下の表は7つのタイプの式ごとに例を示します。

式	タイプ	説明
"こんにちは"	文字	これは文字列定数"こんにちは"です。文字列定数であることを表すために二重引用符が必要です。
"こんにちは"+"みなさん"	文字	二つの文字列"こんにちは"と"みなさん"は＋演算子により結合され"こんにちはみなさん"を返します。
[People]Name+"様"	文字	二つの文字列の結合です。[People]Nameフィールドと文字列"様"が結合されます。フィールドの値が"小林"の場合、"小林様"を返します。
Uppercase ("smith")	文字	この式はUppercase 関数を使用しています。Uppercase 関数は文字列"smith"を英大文字に変換します。そして"SMITH"を返します。
4	数値	これは数値定数 4です。
4 * 2	数値	二つの数値の4と2の乗算です。乗算オペレータの(*)を使用しています。数値の8を返します。.
MyButton	数値	これはボタンの名前（変数名）です。ボタンの現在の値を数値で返します。クリックされた場合に1、それ以外では0（ゼロ）を返します。
!06/12/24!	日付	この式は日付定数で2006年12月24日を表します。
Current date + 30	日付	これは日付を計算しています。"Current date "関数は現在の日付を返します。現在の日付に30日を加えた新しい日付を返します。
?8:05:30?	時間	これは時間定数で、8時5分30秒を表します。
?2:03:04? + ?1:02:03?	時間	これは二つの時間の足し算を行い、3時5分7秒を返します。
True	ブール	これはブール値のTRUE（真）を返します。
10 # 20	ブール	これは二つの数値の論理比較です。#記号は、"等しくない"を表します。10と20は "等しくない"のでTRUE（真）を返します。
"ABC" = "XYZ"	ブール	これは文字列の論理比較です。文字列は等しくないのでFALSE（偽）を返します。
MyPicture + 50	ピクチャ	この式はMy Picture変数に入っているピクチャを右に50ピクセル移動したピクチャを返します。
->[People]Name	ポインタ	この式は[People]Nameフィールドのポインタを返します。
Table (1)	ポインタ	このコマンドは一番目に定義されたテーブルのポインタを返します。

データタイプ

4Dのフィールド、変数、式は以下のデータタイプとなり得ます:

データタイプ	フィールド	変数	式
文字列 (*1)	○	○	○
数値 (*2)	○	○	○
日付	○	○	○
時間	○	○	○
ブール	○	○	○
ピクチャ	○	○	○
ポインタ	×	○	○
BLOB (*3)	○	○	×
配列 (*4)	×	○	×
Integer 64 bits (*5)	○	×	×
Float (*5)	○	×	×
オブジェクト (*6)	×	○	○
未定義	×	○	○

*1. 文字列には、文字フィールド、固定長変数、テキストフィールド、テキスト変数があります。
*2. 数値には、実数、整数、倍長整数のフィールドと変数があります。
*3. BLOBはBinary Large OBjectの省略形です。BLOBの詳細については、DISABLE MENU ITEM の章を参照してください。
*4. 配列にはすべてのタイプの配列が含まれます。詳細については、配列 の章を参照してください。
*5. Integer 64 bitsとFloatタイプは、SQLによってのみ管理されます。4D言語でこれらを使用することは推奨されません。これを行うと、これらのデータタイプは実数に変換され、精度が失われることがあります。
*6. オブジェクトを使用すると、ランゲージによって構築されたデータを操作できるようになります。オブジェクトはは "オブジェクト(ランゲージ)" テーマ内のコマンドによって管理できます。

文字列

文字列とは、以下を示す総称です:

- 文字フィールドまたは変数: 文字フィールドには0～255文字までの文字を格納することができます(上限はフィールドが定義されたときに設定されます)。
- テキストフィールドまたは変数: テキストフィールド、変数、または式には0～2 GBのテキストを格納することができます。
- 任意の文字列またはテキスト式

文字列とテキスト変数には違いはありません。

文字列をテキストフィールドに割り当てることもできますし、その逆も可能です。4Dは必要に応じて変換・切り落としを自動的に行います。式の中で、文字列とテキストは混ぜて使用することが出来ます。

注: このマニュアルでは、コマンド説明における文字列とテキストの引数は特に明記されていない限り、両方とも「文字列」と表記されています。

数値

数値とは、以下を示す総称です。

- 実数のフィールド、変数、または式
- 整数のフィールド、変数、または式
- 倍長整数のフィールド、変数、または式

実数データタイプの範囲は、±1.7e±308 (15桁) です。
整数データタイプ（2バイト整数）の範囲は、-32,768..32,767 (2^15..(2^15)-1)です。
倍長整数データタイプ（4バイト整数）の範囲は、-2^31..(2^31)-1です。

数値データタイプは、異なる数値データタイプに代入することができます。このとき、4Dが必要に応じて変換、切り捨て、丸め処理を行います。ただし、値が範囲外の場合には、変換は正しい値を返しません。数値データタイプは式の中に混在させて使用することができます。

注: このマニュアルでは、実際のデータタイプに関わらず、コマンド説明における実数、整数、倍長整数の引数は特に明記されていない限り、数値と表記されています。

日付

- 日付フィールド、変数、式として認識できる範囲は、100/1/1から32,767/12/31までです。
- 日本語版の4Dを使用した場合、日付の順序は年/月/日の順になります。
- 年を2桁で指定され、30未満の場合2000年代であると解釈されます。30以上の場合は1900年代であると解釈されます。（このデフォルト設定は、SET DEFAULT CENTURYコマンドで変更できます）。

注: このマニュアルでは、コマンド説明における日付引数は特に明記されていない限り、「日付」と表記されています。

JavaScript の日付の変換

JavaScriptにおいて日付はオブジェクトなので、4Dに送られる際には他のオブジェクト同様、JSON フォームを含んだテキストとして送られます。この原理は特に**4D Mobile** または Webエリア を使用する際に適用されます。
JavaScript 日付の JSON フォームは ISO 8601規格に基づき、"2013-08-23T00:00:00Z"のように記述されます。

このテキストは、デベロッパが4Dの日付(C_DATE) へと変換してあげる必要があります。方法は二つあります。:

- JSON Parse コマンドを使用する方法:
  ```
  C_TEXT($1) // 日付をISOフォーマットで受け取る
  C_DATE($d)
  $d:=JSON Parse("\""+$1+"\"";Is Date))
  ```
- Date コマンドを使用する方法:
  ```
  C_TEXT($1) // 日付をISOフォーマットで受け取る
  C_DATE($d)
  $d:=Date($1)
  ```

二つの方法の違いに注意して下さい。JSON Parse コマンドは SET DATABASE PARAMETER を使用して設定された変換モードに従うのに対し、Date コマンドはこれに依存しないという事です。Date コマンドを使用した変換は、常にローカルのタイムゾーンを考慮するということです。

時間

- 時間のフィールド、変数、式の範囲は00:00:00から596,000:00:00までです。
- 日本語版の4Dでは、時間の順序は時:分:秒の順になります。
- 時間は24時間制です。
- 時間の値は数値として扱うことができます。時間から返される数値は時間が表す秒数です。詳細はの節を参照してください。

注: このマニュアルでは、コマンド説明における時間引数は特に明記されていない限り、「時間」と表記されています。

ブール

ブールのフィールド、変数、式は、TRUE（真）またはFALSE（偽）のいずれかになります。

注: このマニュアルでは、コマンド説明におけるブール引数は特に明記されていない限り、「ブール」と表記されています。

ピクチャ

ピクチャのフィールド、変数、式は、任意のWindowsまたはMacintoshのピクチャになります。一般的に、ペーストボード上に置いたり、4Dコマンドやプラグインコマンドを使用してディスクから読み出すことのできる任意のピクチャがこれに含まれます。

注: このマニュアルでは、コマンド説明におけるピクチャ引数は特に明記されていない限り、「ピクチャ」と表記されています。

ポインタ

ポインタの変数または式は、別の変数（配列、配列要素を含む）、テーブルまたはフィールドへの参照です。ポインタタイプのフィールドは、存在しません。

ポインタの詳細については、の節を参照してください。

注: このマニュアルでは、コマンド説明におけるポインタ引数は特に明記されていない限り、「ポインタ」と表記されています。

BLOB

BLOBのフィールドまたは変数は、個別にまたはBLOBコマンドを使用して利用できる一連のバイト列（長さは0バイトから2ギガバイトまで）です。BLOBタイプの式は、存在しません。

注: このマニュアルでは、コマンド説明におけるBLOB引数は特に明記されていない限り、「BLOB」と表記されています。

Object

4D のネイティブなオブジェクトの構造は、よくある「値/名前」というペア(連想配列)に基づいています。これらのオブジェクトの記法は JSON のそれをもとにしていますが、完全に同じというわけではありません。

- プロパティ名は必ずテキストで表現されます(例: "Name")。
- プロパティ値は以下のどれかの型で表現されます:
  - 番号(実数、整数、等)
  - テキスト
  - 配列(テキスト配列、実数配列、ブール配列、オブジェクト配列、ポインター配列)
  - null
  - ブール
  - ポインター(JSON Stringify コマンドを使用して評価されたか、コピーされたときにポインターとして保存されます)
  - 日付("\"YYYY-MM-DDTHH:mm:ssZ\"" というフォーマットで表現されます)
  - オブジェクト(オブジェクトを代入する階層の数に制限はありません)

オブジェクトは オブジェクト(ランゲージ) テーマ内のコマンドによって管理します。

未定義

未定義は、実際にはデータタイプではありません。未定義は、まだ定義されていない変数を指しています。関数（結果を返すプロジェクトメソッド）は、メソッド内で、戻り値 ($0) に未定義式が代入されている場合、未定義値を返すことがあります。フィールドは、未定義にはできません。

配列

配列は、実際にはデータタイプではありません。さまざまな種類の配列（整数配列、テキスト配列等）はこの配列という名前の下にグループ化されています。配列は変数です。配列タイプのフィールドは存在せず、配列タイプの式も存在しません。配列の詳細については、の節を参照してください。

注: このマニュアルでは、コマンド説明における配列引数は特に明記されていない限り、「配列」と表記されています (例: 文字列配列や数値配列など)。

データタイプの変換

4D言語には、データタイプ間の変換を行う演算子やコマンドがあります。4D言語はデータタイプをチェックしています。例えば、"abc"+0.5+!12/25/96!-?00:30:45?のように記述することはできません。これは、シンタックス（構文）エラーになります。

次の表は、基本的なデータタイプ、変換できるデータタイプ、それを実行する際に使用するコマンドを示しています:

データタイプ	文字列に変換	数値に変換	日付に変換	時間に変換
文字列		Num	Date	Time
数値 (*)	String			
日付	String			
時間	String			
ブール		Num		

(*) 時間値は数値として扱うことができます。

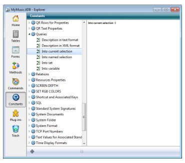
注: この表に示すデータ変換の他に、演算子と他のコマンドを組み合せることで、より洗練されたデータ変換を実行することができます。

定数

定数は、固定値を持つ式です。定数には、名前で選択できる**定義済定数**と実際の値を入力する**リテラル定数**の2種類があります。

定義済定数

4Dではいくつかの**定義済定数**が用意されています。これらの定数はエクスプローラーウィンドウ内でテーマごとにグループ分けがされています。

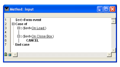

MyMusic.4DB - Explorer
Constants
Home
Tables
Forms
Methods
Commands
Constants
Plug-ins
Trash
QR Rows for Properties
QR Text Properties
Queries
Description in text format
Description in XML format
Into current selection
Into named selection
Into set
Into variable
Relations
Resources Properties
SCREEN DEPTH
SET RGB COLORS
Shortcut and Associated Keys
SQL
Standard System Signatures
System Documents
System Folder
System Format
TCP Port Numbers
Text Values for Associated Stand
Time Display Formats
Into current selection : 0

メソッドエディターウィンドウ内で定義済定数を使用するためには、次のどちらかの手順で行います。

- エクスプローラーウィンドウからメソッドエディターウィンドウへ定数をドラッグ&ドロップしてください。
- メソッドエディターウィンドウに直接定数の名前を入力して下さい。自動補完機構がプログラムのコンテキストに対応した定数を提案します。

定義済定数名は、最大31文字まで指定できます。

「メソッドエディタ」ウインドウや「デバッグ」ウインドウにおいて、定義済定数はデフォルトとして下線付きで表示されます。


Method: Input

```
   $evt=Form event
Case of
   : ($evt=On Load)

   : ($evt=On Close Box)
      CANCEL
End case
```

例えば、上記のウインドウではOn LoadやOn Close Boxが定義済定数です。

リテラル定数

リテラル定数は、次の4つのデータタイプにできます。

- 文字列
- 数値
- 日付
- 時間

文字列定数

文字列定数は、次のように二重引用符("...")で囲んで表します。文字列定数の例を次に示します。

"レコード追加"
"レコードが見つかりません"
"送り状"

空の文字列は、2つの引用符の間に何も入れない状態("")で指定します。

数値定数

数値定数は、実数として記述します。下記に数値定数の例をいくつか次に示します。

27
123.76
0.0076

負の数値は、次のようにマイナス記号(-)を付けて指定します。

-27
-123.76
-0.0076

日付定数

日付定数は、エクスクラメーションマーク(! ... !)で囲んで表します。日付は、"年/月/日"の順で表し、それぞれをスラッシュ(/)で区切ります。
下記に、日付定数の例を示します。

!76/1/1!
!04/4/4!
!06/07/25!

空の日付は、!00/00/0000!のように指定します。

Tip: メソッドエディタでは空の日付を入力するためのショートカットが提供されています。空の日付を入力するには、エクスクラメーションマーク(!)の入力後にenterキーを押します。

注: 2桁の年度は、30以上の場合20世紀(1900年代)、30未満の場合21世紀(2000年代)であると認識します(ただしデフォルト設定が SET DEFAULT CENTURY コマンドを使用して変更されていない場合に限ります)。

時間定数

時間定数は、疑問符(? ... ?)で囲んで表します。
時間は、"時:分:秒"の順で表し、それぞれをコロン(:)で区切ります。時間は24時間制で指定します。

時間定数の例を次に示します。

?00:00:00? ` 午前0時
?09:30:00? ` 午前9時30分
?13:01:59? ` 午後1時1分59秒

空の時間は、?00.00.00?のように指定します。

Tip: メソッドエディタでは空の時間を入力するためのショートカットが提供されています。空の時間を入力するには、疑問符(?)の入力後にenterキーを押します。

変数

4Dのデータは、基本的に全く異なる2つの方法で保持されます。フィールドはディスクに永続的にデータを保存するのに対し、変数はメモリ上に一時的にデータを格納します。

データベースを作成する際、フィールドに名前とデータタイプを指定するのと同様に、変数にも名前とデータタイプを指定します。

以下の変数タイプは、各データタイプと一致します:

- 文字列(*)またはテキスト: 2GBまでの文字列
- 整数: 整数値 -32768 ~ 32767
- 倍長整数: 整数値 -2^31 ~ (2^31)-1
- 実数: ±1.7e±308 (15桁) までの数値
- 日付: 1/1/100 ~ 12/31/32767
- 時間: 00:00:00 ~ 596000:00:00 (24:00までの秒数)
- ブール: True または False
- ピクチャ: 任意のWindowsまたはMacintosh ピクチャ
- BLOB (Binary Large OBject): サイズが2GBまでのバイト列
- ポインタ: ファイル、フィールド、変数、配列、配列要素のポインタ

(*) Unicodeモードでは、文字列とテキストタイプの変数は同じものです。非Unicodeモード (互換モード) では、文字列は255文字までの固定長文字列です。

(ポインタやBLOBを除く) 変数を画面上に表示したり、その中に入力したり、レポートに印刷したりできます。このように、入力可や入力不可のエリア変数はフィールドと同様に振舞います。そしてそれらを作成する際には、フィールドと同じ、組み込みのコントロールを使用できます:

- 表示フォーマット
- 入力フィルタやデフォルト値などのデータ検証
- 文字フィルタ
- 選択リスト (階層リスト)
- 入力可または入力不可値

また、変数は以下のようなこともできます:

- ボタン (ボタン、チェックボックス、ラジオボタン、3Dボタン等) のコントロール
- スライダ (メータ、ルーラ、ダイアル) のコントロール
- スクロールエリア、ポップアップメニュー、ドロップダウンリストのコントロール
- 階層リストや階層ポップアップメニューのコントロール
- ボタングリッド、タブコントロール、ピクチャボタンなどのコントロール
- 保存する必要のない計算結果の表示

変数の作成

変数は、使用することで作成できます。フィールドで行うように、正式にそれらを定義することは必須ではありません。例えば、今日の日付に30日を加えた日付を保持する変数が必要な場合、以下のように書くことができます:

```
MyDate:=Current date+30
```

4DはMyDateを作成し、必要な日付を保持します。コード行は"MyDateに、今日の日付に30日加えたものが代入される"と読むことができます。この後、データベースの必要な個所でMyDateを使用できます。例えば、日付変数を同じタイプのフィールドに格納する場合:

```
[MyTable]MyField:=MyDate
```

しかしながら、通常、変数を明示的に特定のタイプに定義することをお勧めします。コンパイルしようとするデータベースで、変数のタイプを定義する方法については、の節を参照してください。

変数にデータを代入する

変数にデータを格納、または変数からデータをコピーすることができます。変数にデータを置くことを、**変数にデータを代入する**と呼び、それを行うには**代入演算子** (:=) を使用します。代入演算子はフィールドに対してデータを代入する場合にも使用します。

代入演算子は、変数を作成し、変数にデータを代入するために使用します。作成する変数名を代入演算子の左側に書きます。例えば:

```
MyNumber:=3
```

は変数MyNumberを作成し、数値 3を置いています。MyNumberが既に存在すれば、単に数値 3が置かれます。

もちろん、変数からデータを取り出すことができなければ、便利とは言えません。再度代入演算子を使用します。[Products]SizeというフィールドにMyNumber変数の値を代入するには、代入演算子の右側にMyNumberを記述します:

```
[Products]Size:=MyNumber
```

この[Products]Size の値は3になります。この例はとても単純ですが、データをある場所から別の場所へ転送する基本的な方法です。

重要: 代入演算子 (:=) と比較演算子 (=) とを混同しないように注意してください。代入演算子と比較演算子はまったく異なる演算子です。比較演算子に関する詳細は、の節を参照してください。

ローカル、プロセス、およびインタープロセス変数

開発者は**ローカル**、**プロセス**、および**インタープロセス**、3種類の変数の変数を作成できます。この3種類の変数の違いは使用できるスコープにあります。またそれらを使用することのできるオブジェクトも異なります。

ローカル変数

ローカル変数はその名のとおりメソッド内でローカルであり、変数が作成されたメソッド内でのみアクセスすることができ、その他のメソッドからはアク セスできません。メソッド内でローカルであるというのは、「そのスコープがローカルである」という意味です。ローカル変数は、その使用範囲をメソッド内に 限定する場合に用います。

ローカル変数は、以下のような目的のために使用されます:

- 他の変数名との重複を避ける。
- データを一時的に使用する。
- プロセス変数の数を減らす。

ローカル変数の名前は必ずドル記号 ($) で始め、この記号を除く31文字までの文字を指定できます。これより長い名前を指定すると、4Dは余分の32文字以降を切り捨てます。

多くのメソッドや変数を持つデータベースで作業する場合、現在作業しているメソッド内でのみ使用可能な変数を必要とする場合がよくあります。この場合、同じ変数名が他で使用していないかどうかを気にすることなくローカル変数を作成することができます。

しばしばデータベースで、ユーザからの小さなデータを必要とする場合があります。Request関数は、ユーザから1つのデータを取得します。Request関数はデータの入力を求めるダイアログボックスを表示し、ユーザがデータを入力した時点で、その入力情報を返します。通常このデータはメソッド内で長期間必要となることはありません。これは、ローカル変数を使用する良い例です。

```
$vsID:=Request("Please enter your ID:")
If(OK=1)
   QUERY([People];[People]ID =$vsID)
End if
```

このメソッドは、ユーザに従業員IDを入力するように要求します。ローカル変数$vsIDにレスポンスが代入され、ユーザが入力したIDに基づき検索が行われます。このメソッドが終了した時点で、$vsIDローカル変数はメモリから消去されます。この変数は1回のみ、またこのメソッド内でのみしか必要としないためローカル変数で十分です。

プロセス変数

プロセス変数は、プロセス内でだけ使用可能です。この変数はプロセスメソッドと、そのプロセス内で呼び出された他のメソッドで使用することができます。

プロセス変数には名前の前に付ける接頭辞がありません。プロセス変数名の長さは、最大31文字まで指定できます。

インタプリタモードでは、変数は動的にメモリ上に作成、消去されます。これに対してコンパイルモードでは、作成したすべてのプロセス (ユーザプロセス) で同じプロセス変数定義が共有されますが、変数のインスタンスはプロセス毎に異なるものとなります。例えばプロセス変数myVarは、プロセスP_1とプロセスP_2の両方に存在しますが、それぞれ異なるインスタンスとなります。

バージョン6より、GET PROCESS VARIABLEやSET PROCESS VARIABLEを使用して、あるプロセスから他のプロセスのプロセス変数の値を取り出したり、設定したりできるようになりました。これらのコマンドの利用は、以下のような状況に限定することが、良いプログラミングの作法です:

- 特定の個所あるいはコードにおけるプロセス間通信
- プロセス間のドラッグ＆ドロップ処理
- クライアント/サーバにおいて、クライアントマシンプロセスとサバマシン上のストアドプロシージャプロセス間通信

詳細はの章を参照してください。

インタープロセス変数

インタープロセス変数はデータベース全体で使用することができ、すべてのプロセスで共有されます。主としてプロセスの間で情報を共有するために使われます。

インタープロセス変数の名前は、必ずインタープロセス記号（<>）で始めます。記号の後に31バイトまでの名前を指定できます。

注: このシンタックスはWindowsとMacintosh両方で使用できます。

クライアント/サーバでは、各マシン（クライアントマシンとサーバマシン）で同じインタープロセス変数定義を共有しますが、マシンごとに各変数のインスタンスが存在します。

フォームオブジェクト変数

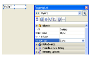
フォームエディタで、ボタン、ラジオボタン、チェックボックス、スクロールエリア、サーモメータ等のアクティブオブジェクトに変数名を与えると、同じ名前の変数が自動的に生成されます。例えば、**MyButton**という名前のボタンを作成すると、**MyButton**という名前の変数が作成されます。この変数名はボタンのラベルではなく、ボタンの名前であることに注意してください。

フォームオブジェクト変数を使用して、オブジェクトの制御やモニタを実行できます。例えば、このボタンがクリックされると、その変数に1が設定され ます。それ以外の場合はすべて0になります。サーモメータやダイアルに関連付けられた変数では、現在の値を読んだり、あるいは値を変更することができます。例えばサーモメータをドラッグすると、変数の値は新しい設定値に変更されます。同様にメソッドで変数の値を変更すると、サーモメータは新しい値を表現 するために再描画されます。

フォームと変数のより詳しい情報は、4D Design Referenceマニュアルやの節を参照してください。

ダイナミック変数

(ボタンや入力可能変数、チェックボックス等の) フォームオブジェクトに割り当てる変数を、必要に応じて4Dに動的に作成させることができます。これを行うには、プロパティリストの"変数名"フィールドを空にします:



変数名が与えられていないと、フォームがロードされたとき、4Dはインタープリターのプロセス変数の空間内でユニークな名前を計算し、その名前でオブジェクト用に新しい変数を作成します (このメカニズムはコンパイルモードでも使用することができます)。この一時的な変数はフォームが閉じられるときに破棄されます。
この方式をコンパイルモードで動作させるために、ダイナミック変数の型を明示的に指定しなければなりません。これを行うには2つの方法があります:

- プロパティリストの"変数タイプ"を使用して型を指定する。
  注: 変数に名前が与えられている場合、"変数タイプ"は実際には変数の型を指定しません。プロパティリストのオプション表示を更新するだけです (ピクチャー変数を除く)。名前付き変数の型を指定するには、コンパイラテーマのコマンドを使用しなければなりません。
- フォームがロードされるときにVARIABLE TO VARIABLE等を使用する特別な初期化コードを実行する。

```
 If(Form event=On Load)
    C_TEXT($init)
    $Ptr_object:=OBJECT Get pointer(Object named;"comments")
    $init:=""
    VARIABLE TO VARIABLE(Current process;$Ptr_object->;$init)
 End if
```

注: ダイナミック変数を指定し (名前なし変数を作成し)、"変数タイプ"でなしを選択、そして初期化コードを実行しない場合、コンパイラが型指定エラーを返します。

4Dコード中で、ダイナミック変数にはOBJECT Get pointerコマンドで取得できるポインターを通してアクセスできます。例えば:

```
  // "tstart"オブジェクトの変数に時刻12:00:00を代入する
 $p :=OBJECT Get pointer(Object named;"tstart")
 $p->:=?12:00:00?
```

このメカニズムを使用する利点は2つあります:

- ひとつのホストフォーム上で複数個配置することの可能なサブフォームタイプのコンポーネント開発を可能にします。例えば開始日と終了日を表す日付ピッカーオブジェクトをホストフォーム上に配置するケースを考えてみましょう。このサブフォームでは日付を選択するためのオブジェクトが使用されます。このオブジェクトは開始日と終了日それぞれで異なる日付を選択できなければなりません。4Dにダイナミック変数を生成させることでユニークな変数を得ることができ、この問題を解決できます。
- またメモリの利用を減少させることができます。フォームオブジェクトではプロセス変数とインタープロセス変数しか使用できません。しかしコンパイルモードでは、各プロセス変数のインスタンスが (サーバープロセスを含め) すべてのプロセスに対して作成されます。このインスタンスはセッション中にフォームが使用されない場合でもメモリを消費します。フォームのロード時、4Dにダイナミック変数を作成させることで、メモリを節約できます。

注: 変数名が指定されていない場合、フォームエディター上にはクォーテーションマークで囲まれたオブジェクト名が表示されます (デフォルトでオブジェクトに変数名が表示される場合)。

システム変数

4Dは、**システム変数**と呼ばれる多くの変数を保守します。これらの変数を使用して、多くの処理をモニタすることができます。すべてのシステム変数がプロセス変数であり、プロセスの中でのみアクセスできます。

最も重要なシステム変数は**OK**システム変数です。名前が示すように、特定のプロセスで、何かの事象がOKかどうかを通知します。例え ば、レコードは保存されたか、重要な処理が終了したか、ユーザがOKボタンをクリックしたか等を判断できます。システム変数OKは、処理が正常に完了した 場合1が設定され、そうでない場合0が設定されます。

システム変数に関する詳細は、の節を参照してください。

システム変数

4Dが管理する**システム変数**を使用して、異なる動作の実行をコントロールできます。すべてのシステム変数はプロセス変数であり、ひとつのプロセス内でのみアクセスできます。この節では4Dのシステム変数について説明します。

これらの変数の型に関する情報は型指定ガイドのシステム変数を参照してください。

OK

システム変数OKは最も頻繁に使用されます。通常、処理が成功すると1が代入され、成功しなかったときに0が代入されます。多くの4Dコマンドはシステム変数OKの値を変更します。各コマンドの説明を参照し、コマンドがこのシステム変数に影響を与えるかどうかを調べてください。

このドキュメントでは、アイコンOKは、そのコマンドがOK変数の値を更新することを示します。
このアイコンをクリックすると、関連するコマンドのリストを表示させることができます。

OK変数を更新するコマンドの一覧を表示するにはここをクリックして下さい。

Document

Documentシステム変数には、以下のコマンドを使用して最後に開かれたり作成されたりしたファイルのパス名 (アクセスパス+ファイル名) が格納されます:

- Append document
- BUILD APPLICATION
- Create document
- Create resource file
- EXPORT DATA
- EXPORT DIF
- EXPORT SYLK
- EXPORT TEXT
- IMPORT DATA
- IMPORT DIF
- IMPORT SYLK
- IMPORT TEXT
- GET DOCUMENT ICON
- LOAD SET
- LOAD VARIABLES
- Open document
- Open resource file
- PRINT LABEL
- QR REPORT
- READ PICTURE FILE
- SAVE VARIABLES
- SAVE SET
- Select document
- SELECT LOG FILE
- SET CHANNEL
- USE ASCII MAP
- WRITE PICTURE FILE

FldDelimit

システム変数FldDelimitは、テキストを読み込んだり書き出したりする際に、フィールドの区切りとして使用する文字のコードが格納されています。デフォルトの値はタブに相当する9です。区切り文字を変更する場合は、FldDelimitの値を変更してください。

RecDelimit

システム変数RecDelimitは、テキストを読み込んだり書き出したりする際に、レコードの区切り文字として使用する文字のコードが格納されています。デフォルトの値はキャリッジリターン(CR)に相当する13です。区切り文字を変更する場合は、RecDelimitの値を変更してください。

Error, Error method, Error line

これらの変数はON ERR CALLコマンドでインストールされたエラー処理メソッド内でのみ使用できます。エラーの原因となったメソッド内でこれらの値を参照したい場合、自身のプロセス変数にコピーしてください。

- Error: 倍長整数型のシステム変数です。この変数はエラーコードを格納します。4Dおよびシステムのエラーコードはテーマでリストされています。
- Error method: テキスト型のシステム変数です。この変数にはエラーの発生元となったメソッド名が格納されます。
- Error line: 倍長整数型のシステム変数です。この変数にはエラー発生元のメソッドの、エラーが発生した行番号が格納されます。

MouseDown、MouseX、MouseY、KeyCode、ModifiersおよびMouseProc

これらのシステム変数は、ON EVENT CALLコマンドでインストールされたメソッドの中でのみ参照できます。

- システム変数MouseDownは、マウスのボタンが押されたときに1が、それ以外の場合は0に代入されます。
- イベントがMouseDownの時(MouseDown=1)、システム変数MouseXとMouseYにはマウスがクリックされた場所の水平座標と垂直座標が代入されます。両方の値ともピクセル単位で表わされ、ウインドウのローカルな座標システムを使用します。

注: ピクチャフィールドや変数がクリックされると、On ClickedやOn Double clicked、およびOn Mouse EnterとOn Mouse Move内で、MouseXとMouseYシステム変数にクリックのローカル座標が返されます。詳細はおよびSVG Find element ID by coordonatesの節を参照してください。

- システム変数KeyCodeには、押されたキーのASCIIコードが代入されます。押されたキーがファンクションキーの場合、KeyCodeには特殊コードがセットされます。文字コードとファンクションキーコードについては、Unicode コード、Ascii コードおよびの節で説明します。
- システム変数Modifiersは、キーボードのモディファイアキーの値を含んでいます(Ctrl/command、Alt/option、Shift、CapsLock)。システム変数Modifiersは、ON EVENT CALLコマンドによってインストールされた"割り込み処理"においてのみ意味を持ちます。
- システム変数MouseProcは、イベントが発生したプロセス番号を含みます。

ポインタ

説明

ポインタは、（プログラミングにおける）データを参照するための高度な方法です。

プログラミング言語を使用する場合、名前を用いてテーブル、フィールド、変数、配列等をアクセスします。通常は、名前によるデータの参照が最も簡単な方法ですが、名前を使用しないでデータを参照する、またはアクセスした方が便利な場合もあります。この場合、ポインタを使用すると実現できます。

ポインタの背景にある概念は、日常生活でもよく用いられています。対象物を正確に知らないで何かを示すことがあります。例えば、友人に対して"登録番号123ABDの車に乗ろう"と言わないで"君の車に乗ろう"という場合があります。つまり、"登録番号123ABDの車"を"君の車"で示したわけです。この場合、"登録番号123ABDの車"はオブジェクトの名前で、"君の車"はオブジェクトを参照するためのポインタと考えることができます。

あるものの対象物を明示しないで参照することができると非常に便利です。例えば友人が新しい車に買い替えても、同じく"君の車"と言うことができます。ポインタも同じように機能します。例えば、ある場合は数値フィールド"Age"を参照したポインタで、別の場合には数値変数"Old Age"を参照することもできます。この場合のポインタは、計算に使用する数値データを参照しています。

テーブル、フィールド、変数、配列、配列要素を参照するためにポインタを使用することができます。以下の表に、各タイプの例を示します。

オブジェクト	参照する	使用する	割り当てる
テーブル	vpTable:=->[Table]	DEFAULT TABLE(vpTable->)	n/a
フィールド	vpField:=->[Table]Field	ALERT(vpField->)	vpField->:="John"
変数	vpVar:=->Variable	ALERT(vpVar->)	vpVar->:="John"
配列	vpArr:=->Array	SORT ARRAY(vpArr->;>)	COPY ARRAY (Arr;vpArr->)
配列要素	vpElem:=->Array{1}	ALERT (vpElem->)	vpElem->:="John"

ポインタの使用例

例題を用いてポインタの使用方法について説明します。以下の例は、ポインタを通して変数にアクセスする方法を示しています。まず、変数の作成から始めます。

```
MyVar:="Hello"
```

"MyVar"は、文字列"Hello"を含む変数です。"MyVar"に対するポインタを作成します。

```
MyPointer:=->MyVar
```

ポインタ記号（->）は、"･･･に対するポインタを求める"ことを意味します。ここでは、"MyVar"を参照するポインタを呼び出します。このポインタは、代入演算子（:=）で"MyPointer"に対して割り当てられます。

"MyPointer"は、"MyVar"に対するポインタを含む変数です。"MyPointer"は、"Hello"という"MyVar"の値を含みませんが、"MyVar"に含まれる値を指すことはできます。以下の式は"MyVar"の値を返します。

```
MyPointer->
```

前述の式は、"Hello"という文字列を返します。ポインタ記号（->）をポインタの後につけると、そのポインタの指すオブジェクトの値を示します。

ポインタ記号（->）を後につけたポインタを使用してオブジェクトを参照することの意味を理解することが重要です。つまり、変数"MyVar"を使用することと、式"MyPointer->"を使用することは、全く同じ意味です。

例えば、以下のステートメントはアラートボックスに文字列"Hello"を表示します。

```
ALERT(MyPointer->)
```

"MyPointer"を使用して"MyVar"の値を変更することもできます。下記のステートメントは、変数"MyVar"に文字列"Goodbye"を代入します。

```
MyPointer->:="Goodbye"
```

この2つの"MyPointer->"を使用したステートメントのとおり、"MyVar"を使用した場合と全く同じ動作を実行します。以下の2つのステートメントも、全く同じ動作を実行します。両方とも、変数"MyVar"の現在の値をアラートボックスに表示します。

```
ALERT(MyPointer->)
ALERT(MyVar)
```

以下の2つのステートメントも、全く同じ動作を実行します。両方とも"MyVar"に、文字列"Goodbye"を代入します。

```
MyPointer->:="Goodbye"
MyVar:="Goodbye"
```

ボタンへのポインタを使用する

この節では、ボタンを参照するためのポインタの使用方法について説明します。ボタンは（プログラミング言語の観点から）、変数に属します。この節では、ボタンを参照するためのポインタの例を使用していますが、ここで示す概念は、ポインタで参照できる他のすべての種類のオブジェクトでも同様です。

フォーム上に無効にしたり有効にしたりを切り替える多数のボタンがあります。それぞれのボタンにはTRUEまたはFALSEの状態が割り当てられます。その状態に基づき、ボタンの有効と無効を切り替えます。ボタンの有効/無効を切り替えるたびに以下のテストを行います。

```
OBJECT SET ENABLED(MyButton;Condition)
```

フォーム中のその他のボタンに対しても、同様の判定を実行する必要があります。さらに効率良く判定処理を実行するためには、各ボタンの参照にポインタを用いて判定を実行するサブルーチンを使用します。

サブルーチンでボタン変数を参照するには、ポインタを使用することができます (他にオブジェクト名を参照する方法もありますが、ここではポインタを使用する方法を紹介します)。例えば、以下のプロジェクトメソッドSet Buttonはポインタを使用してボタンを参照しています。

```
  // SET BUTTON プロジェクトメソッド
  // SET BUTTON ( Pointer ; Boolean )
  // SET BUTTON ( -> Button ; Enable or Disable )

  // $1: ボタンのポインタ
  // $2: Boolean. もしTRUEならボタンを使用可とする。;もしFALSEならボタンを使用不可とする。

OBJECT SET ENABLED($1->;$2)
```

プロジェクトメソッド"Set Button"は以下のように呼び出します。

```
  ` ...
SET BUTTON(->bValidate;True)
  ` ...
SET BUTTON(->bValidate;False)
  ` ...
SET BUTTON(->bValidate;([Employee]Last Name#""))
  ` ...
For($vlRadioButton;1;20)
   $vpRadioButton:=Get pointer("r"+String($vlRadioButton))
   SET BUTTON($vpRadioButton;False)
End for
```

テーブルへのポインタを使用する

プログラミング言語でテーブルの代わりにポインタを使用することができます。
以下のようなステートメントで、テーブルのポインタを作成します。

```
TablePtr:=->[anyTable]
```

あるいは、以下のようにTableコマンドを使用してテーブルのポインタを得ることができます。

```
TablePtr:=Table(20)
```

以下のようにコマンドに対して、ポインタを指定することができます。

```
DEFAULT TABLE(TablePtr->)
```

フィールドへのポインタを使用する

言語がフィールドを期待する場所ではどこでも、フィールドを参照するためにポインタの逆参照を使用できます。以下のようなステートメントで、フィールドのポインタを作成します。

```
FieldPtr:=->[aTable]ThisField
```

あるいは、以下のようにFieldコマンドを使用してフィールドのポインタを得ることができます。

```
FieldPtr:=Field(1;2)
```

以下のようにコマンドに対して、逆参照したポインタを指定することができます。

```
OBJECT SET FONT(FieldPtr->;"Osaka")
```

変数へのポインタを使用する

この節の最初の例は変数へのポインタの使用を説明しています。

```
MyVar:="Hello"
MyPointer:=->MyVar
```

ポインタは、インタープロセス変数、プロセス変数の他、バージョン2004.1からはローカル変数にも利用できるようになりました。

プロセス変数もしくはローカル変数にポインタを使う場合、参照される変数はポインタが使用される時点で既に定義されていなければなりません。
ロー カル変数は、それらを作成したメソッドの実行が終わると破棄され、プロセス変数もそれを作成したプロセスの終了時に削除される点に留意してください。存在 しない変数をポインタが呼び出そうとすると、インタープリタモードでは（「変数が設定されていません」という内容の）シンタックスエラーが起きます。コン パイルモードでは、さらに重大なエラーが発生する可能性があります。

Note: ローカル変数について：ローカル変数のポインタを使用すると、プロセス変数の使用を控えることができます。ポインタは、同じプロセス内のローカル変数にのみ使用することができます。
ポインタの参照先が、別のメソッドで宣言されたローカル変数の場合、これをデバッガに表示すると、オリジナルのメソッド名が、ポインタの後の括弧内に表示されます。例としてMethod1で以下のように書いたとします。

```
$MyVar:="Hello world"
Method2(->$MyVar)
```

Method2で、デバッガは$1を次のように表示します。

$1 ->$MyVar (Method1)

$1の値は、次のようになります。
$MyVar (Method1) "Hello world"

配列要素へのポインタを使用する

配列要素に対するポインタを作成することができます。以下の例は配列を作成し、配列の最初の要素を指し示すポインタを変数ElemPtrに割り当てます。

```
ARRAY REAL(anArray;10) ` 配列を作成
ElemPtr:=->anArray{1} ` 配列要素へのポインタを作成
```

以下のように、ポインタの参照先である配列要素に値を代入することができます。

```
ElemPtr->:=8
```

配列へのポインタを使用する

配列に対するポインタを作成することができます。以下の例は配列を作成し、配列を指し示すポインタを変数"ArrPtr"に割り当てます。

```
ARRAY REAL(anArray;10) ` Create an array
ArrPtr:=->anArray ` Create a pointer to the array
```

ポインタが配列をポイントしていることを理解することが重要です。配列要素ではありません。例えば、ポインタの逆参照を以下のように使用できます:

```
SORT ARRAY(ArrPtr->;>) ` 配列のソート
```

配列のポインタを使用し、例えば4番目の要素にアクセスするには、以下のように記述します。

```
ArrPtr->{4}:=84
```

ポインタ配列の使用

関連するオブジェクトのグループを参照する場合にポインタ配列を使用すると便利な場合があります。

そのようなオブジェクトのグループの例として、フォーム上の変数のグリッドがあります。グリッドのそれぞれの変数には、Var1"、"Var2"、. . . 、"var10"という連番が付いているとします。間接的にこれらの変数を数字で参照することができます。ポインタの配列を作成し、各変数を指すためにポ インタを初期化すれば、変数を簡単に参照することができます。例えば、配列を作成し各要素を初期化するために、以下のようなステートメントを使用します。

```
ARRAY POINTER(apPointers;10) ` 10の要素を持つポインタ配列をを作成する。
For($i;1;10) `各変数に対して1回ずつループする
   apPointers{$i}:=Get pointer("Var"+String($i)) `配列要素を初期化する。
End for
```

Get pointer 関数は、指定された変数名で示されるオブジェクトへのポインタを返します。

いくつかの変数への参照が必要な場合は、配列要素を使用します。例えば、変数に10個の日付を代入する場合は（変数が日付タイプであると想定して）、以下のように記述します。

```
For($i;1;10) ` 各変数に対して1回ずつループする
   apPointers{$i}->:=Current date+$i  ` 日付を代入
End for
```

ポインタを使用したボタンの設定

フォーム中に関連する一連のラジオボタンがある場合に、それらを素早く設定しなければならないことがあります。名前を使用して各ラジオボタンを直接参照することは、効率が良くありません。
今ここにグループ化された、Button1、Button2、… 、Button5という名前の5つのラジオボタンがあったとします。

一連のラジオボタンでは、1つのラジオボタンのみがオンになります。オンになっているラジオボタンの番号は、数値フィールドに記憶されます。例えば、[Preferences]Settingの値が3の場合は、Button3に1を設定します。フォームメソッドは以下のようなプログラムにより、ボタンをセットします。

```
Case of
   :(Form event=On Load)
  ` ...
      Case of
         :([Preferences]Setting=1)
            Button1:=1
         :([Preferences]Setting=2)
            Button2:=1
         :([Preferences]Setting=3)
            Button3:=1
         :([Preferences]Setting=4)
            Button4:=1
         :([Preferenca]Setting=5)
            Button5:=1
      End case
  ` ...
End case
```

上記例では各ラジオボタンごとに判定しなければなりません。フォーム内に多くのラジオボタンがあると、非常に長いメソッドになる場合もあります。この問題を解決するために、ポインタを使用します。

Get pointer 関数を使用して、ラジオボタン（あるいは任意のボタン）のポインタを作成することができます。値を設定する必要のあるラジオボタンを参照するためにポインタを使用します。次に改善されたメソッドの例を示します。

```
Case of
   :(Form event=On Load)
  ` ...
      $vpRadio:=Get pointer("Button"+String([Preferences]Setting))
      $vpRadio->:=1
  ` ...
End case
```

ラジオボタンの番号が、"設定"フィールドに記憶される必要があります。これは、フォームメソッドでOn Clickedイベントに対し以下の行を加えると実現できます。

```
[Preferences]Setting:=Button1+(Button2*2)+(Button3*3)+(Button4*4)+(Button5*5)
```

メソッドにポインタを渡す

ポインタを引数としてメソッドに渡すことができます。メソッド内で、ポインタによって参照されるオブジェクトを修正することができます。
例えば、以下のメソッド"Take Two"は、引数に2つのポインタ持ちます。最初の引数は、大文字に変換するオブジェクトを参照します。2番目の引数は小文字に変換するオブジェクトを参照します。次に、そのメソッドを示します。

```
  ` TAKE TWO project method
  ` $1: Pointer to a string field or variable. Change this to uppercase.
  ` $2: Pointer to a string field or variable. Change this to lowercase.
$1->:=Uppercase($1->)
$2->:=Lowercase($2->)
```

以下のステートメントは、フィールド値を大文字に、変数を小文字に変更するためにメソッド"Take Two"を使用します。

```
TAKE TWO(->[My Table]My Field;->MyVar)
```

このフィールド[MyTable]MyFieldが"jones"であれば、"JONES"に変更されます。一方変数MyVarが"HELLO"であれば、"hello"に変更されます。
メソッドTake Twoで使用するポインタと、参照されるオブジェクトのデータタイプが一致していることが重要です。この例では、ポインタに対して必ず文字型またはテキスト型のオブジェクトを割り当てなければなりません。

ポインタのポインタ

より高度な使い方として、他のポインタを参照するためにポインタを使うことができます。以下の例を考察してください。

```
MyVar:="Hello"
PointerOne:=->MyVar
PointerTwo:=->PointerOne
(PointerTwo->)->:="Goodbye"
ALERT((Point Two->)->)
```

この例はアラートボックスに"Goodbye"を表示します。

各行について見ていきましょう。

- MyVar:="Hello"
  --> この行は、変数MyVarに"Hello"の文字列を代入しています。
- PointerOne:=->MyVar
  --> ポインタ変数PointerOne に、変数MyVarのポインタを代入します。
- PointerTwo:=->PointerOne
  --> 新たなポインタ変数PointerTwoにPointerOne変数のポインタを代入します。
- (PointerTwo->)->:="Goodbye"
  --> "PointerTwo->"は、変数PointerOneを示します。"(PointerTwo->)->"は、変数MyVarを示します。結果として、文字列"Goodbye"は、変数MyVarに代入されます。
- ALERT ((PointerTwo->)->)
  --> 先の説明と同様に"(PointerTwo->)->"は変数MyVarを示すので、結果としてアラートボックスには変数MyVarの内容が表示されます。

以下の例では、変数MyVarに"Hello"が代入されます。

```
(PointerTwo->)->:="Hello"
```

以下の例では、変数NewVarに変数MyVarの値である"Hello"が代入されます。

```
NewVar:=(PointerTwo->)->
```

重要：複数の参照には括弧が必要です。

識別子

本節は、4D言語のさまざまなオブジェクトを命名するための規則を説明します。すべてのオブジェクトの名前は、次の規則に従います:

- 名前は半角アルファベットまたはアンダースコアー (_)文字で始めます。
- その後、名前には半角アルファベット文字、数字、スペース、アンダースコアを使用できます。
- ピリオド (.)、スラッシュ (/)、クォーテーションマーク (")、コロン (:) は使用できません。
- +や*等、演算子に用いられる文字は使用できません。
- 名前の最後につけたスペースは無視されます。

Note: オブジェクトがSQLで処理される場合には、追加のルールに従う必要があります。文字 _0123456789abcdefghijklmnopqrstuvwxyz のみを使用できます。また、名前にはコマンドや属性などのSQLキーワードを含めることができません。ストラクチャエディタのインスペクタ下部にある"SQL"エリアには、テーブル名やフィールド名として許可されない文字があると警告が表示されます。

テーブル

角カッコ内 ([...]) に名前を入れると、テーブルを表します。テーブル名は、31文字以内で指定します。

例題

```
DEFAULT TABLE([Orders])
FORM SET INPUT([Clients];"Entry")
ADD RECORD([Letters])
```

フィールド

フィールドが属するテーブルを最初に指定することで、フィールドを表します。フィールドの名前はテーブル名のすぐ後に続けます。フィールド名は31文字以内で指定します。

例題

```
[Orders]Total:=Sum([Line]Amount)
QUERY([Clients];[Clients]Name="Smith")
[Letters]Text:=Capitalize text([Letters]Text)
```

インタープロセス変数

名前の先頭にインタープロセス (<>) 記号を付けることによって、インタープロセス変数を表します。インタープロセス変数名は、インタープロセス (<>) 記号を除いて31文字以内で指定します。

例題

```
<>vlProcessID:=Current process
<>vsKey:=Char(KeyCode)
If(<>vtName#"")
```

プロセス変数

名前 (<>記号や$記号から始まらない) を使用して、プロセス変数を表します。プロセス変数名は、31文字以内で指定します。

例題

```
vrGrandTotal:=Sum([Accounts]Amount)
If(bValidate=1)
   vsCurrentName:=""
```

ローカル変数

ドル記号 ($) を名前の先頭につけてローカル変数を表します。ローカル変数名は、ドル ($) 記号を除いて31文字以内で指定します。

例題

```
For($vlRecord;1;100)
   If($vsTempVar="No")
      $vsMyString:="Hello there"
```

配列

名前を使用して、配列を表します。これは配列作成時に配列宣言コマンド (ARRAY LONGINT等) に渡す名前です。配列は変数であり、スコープに基づいて次の3種類があります:

- インタープロセス配列
- プロセス配列
- ローカル配列

インタープロセス配列
インタープロセス配列の名前は、名前の先頭にインタープロセス (<>) 記号が付きます。インタープロセス配列名は、インタープロセス (<>) 記号を除いて31文字以内で指定します。

例題

```
ARRAY TEXT(<>atSubjects;Records in table([Topics]))
SORT ARRAY(<>asKeywords;>)
ARRAY INTEGER(<>aiBigArray;10000)
```

プロセス配列
(<>記号または$記号から始まらない) 名前を使用して、プロセス配列を表わします。プロセス配列名は31文字以内で指定します。

例題

```
ARRAY TEXT(atSubjects;Records in table([Topics]))
SORT ARRAY(asKeywords;>)
ARRAY INTEGER(aiBigArray;10000)
```

ローカル配列
配列名がドル記号 ($) で始まるものは、ローカル配列です。ローカル配列名は、ドル ($) 記号を除いて31文字以内で指定します。

例題

```
ARRAY TEXT($atSubjects;Records in table([Topics]))
SORT ARRAY($asKeywords;>)
ARRAY INTEGER($aiBigArray;10000)
```

配列の要素
中カッコ ({…}) を使用して、インタープロセス配列、プロセス配列、ローカル配列の要素を参照します。参照される配列要素は数式で表されます。

例題

```
 ` インタープロセス配列の要素を指定する
If(<>asKeywords{1}="Stop")
   <>atSubjects{$vlElem}:=[Topics]Subject
   $viNextValue:=<>aiBigArray{Size of array(<>aiBigArray)}

 ` プロセス配列の要素を指定する
   If(asKeywords{1}="Stop")
      atSubjects{$vlElem}:=[Topics]Subject
      $viNextValue:=aiBigArray{Size of array(aiBigArray)}

 ` ローカル配列の要素を指定する
      If($asKeywords{1}="Stop")
         $atSubjects{$vlElem}:=[Topics]Subject
         $viNextValue:=$aiBigArray{Size of array($aiBigArray)}
```

二次元配列の要素
中カッコ ({…}) を2回使用して、2次元配列の要素を参照します。参照される要素は2組の中カッコ内の2つの数式で表されます。

例題

```
 ` 2次元インタープロセス配列の要素を指定する
If(<>asKeywords{$vlNextRow}{1}="Stop")
   <>atSubjects{10}{$vlElem}:=[Topics]Subject
   $viNextValue:=<>aiBigArray{$vlSet}{Size of array(<>aiBigArray{$vlSet})}

 ` 2次元プロセス配列の要素を指定する
   If(asKeywords{$vlNextRow}{1}="Stop")
      atSubjects{10}{$vlElem}:=[Topics]Subject
      $viNextValue:=aiBigArray{$vlSet}{Size of array(aiBigArray{$vlSet})}

 ` 2次元ローカル配列の要素を指定する
      If($asKeywords{$vlNextRow}{1}="Stop")
         $atSubjects{10}{$vlElem}:=[Topics]Subject
         $viNextValue:=$aiBigArray{$vlSet}{Size of array($aiBigArray{$vlSet})}
```

フォーム

フォームの名前は文字列を使用して表します。フォーム名は31文字以内で指定します。

例題

```
FORM SET INPUT([People];"Input")
FORM SET OUTPUT([People];"Output")
DIALOG([Storage];"Note box"+String($vlStage))
```

メソッド

名前を使用して、メソッド (プロシージャおよび関数) を表します。メソッド名は31文字以内で指定します。

Note: 結果を返さないメソッドは**プロシージャ**とも呼ばれます。結果を返すメソッドを**関数**と呼びます。

例題

```
If(New client)
   DELETE DUPLICATED VALUES
   APPLY TO SELECTION([Employees];INCREASE SALARIES)
```

Tip: 4Dの組み込みコマンドと同じ命名規約を利用することは良いプログラミングテクニックです。メソッド名には大文字を使用しますが、メソッドが関数の場合、メソッド名の最初の文字だけを大文字にします。このように命名することにより、数ヶ月後に保守のためデータベースを再度開いたときに、エクスプローラウィンドウでその名前を見ただけで、メソッドが結果を返すかどうかがわかります。

Note: メソッドを呼び出すには、メソッド名を入力するだけです。しかしON EVENT CALL等4Dの組み込みコマンドの一部やプラグインコマンドは、引数のメソッド名を文字列 (ダブルクォートで括る) として渡すものがあります。

例題

```
 ` このコマンドはメソッドを (関数) またはフォーミュラを期待する
QUERY BY FORMULA([aTable];Special query)
 ` このコマンドはメソッドを (プロシージャ) または文を期待する
APPLY TO SELECTION([Employees];INCREASE SALARIES)
 ` このコマンドはメソッド名を期待する
ON EVENT CALL("HANDLE EVENTS")
 ` このプラグインコマンドはメソッド名を期待する
WR ON ERROR("WR HANDLE ERRORS")
```

メソッドに引数を渡すことができます。引数はメソッド名の後のカッコ内に記述します。各引数は、セミコロン (;) で区切ります。引数は、呼び出されたメソッド内で、連番付きのローカル変数$1, $2,…, $nとして使用できます。さらに、複数の連続する引数は、${n}というシンタックスを用いて使用できます。nは数値で引数の番号を示します。

関数の戻り値は、ローカル変数$0に代入することで指定します。

例題

```
 ` DROP SPACESの中で、$1はフィールド[People]Nameへのポインタ
DROP SPACES(->[People]Name)

 ` Calc creatorの中で:
 ` - $1は数値の1
 ` - $2は数値の5
 ` - $3はテキストまたは文字列の"Nice"
 ` - 戻り値は$0に代入される
$vsResult:=Calc creator(1;5;"Nice")

 ` Dumpの中で:
 ` - 3つの引数はテキストまたは文字列
 ` - これらの引数は$1, $2, $3で参照できる
 ` - またこれらの引数を${$vlParam}でも参照できる ($vlParamは1, 2, 3)
 ` - 戻り値は$0に代入される
vtClone:=Dump("is";"the";"it")
```

プラグインコマンド (外部プロシージャ、関数、そしてエリア)

プラグインで定義された名前を使用して、プラグインコマンドを表します。プラグインコマンド名は31文字以内で指定します。

例題

```
WR BACKSPACE(wrArea;0)
$pvNewArea:=PV New offscreen area
```

セット

スコープに基づき、2つのタイプのセットがあります:

- インタープロセスセット
- プロセスセット

4D Serverには以下もあります:

- クライアントセット

インタープロセスセット
インタープロセスセットの名前は、名前の先頭にインタープロセス (<>) 記号が付きます。インタプロセスセット名は、インタープロセス (<>) 記号を除いて255文字以内で指定します。

プロセスセット
セットの名前を表す文字列式を使用してプロセスセットを表します (<>記号も$記号も名前の先頭につきません) 。プロセスセット名は、255文字以内で指定します。

クライアントセット
クライアントセット名は、名前の先頭にドル ($) 記号を指定します。クライアントセット名は、ドル記号を除いて255文字以内で指定します。

Note: セットはサーバマシン上で保守されます。効率や特殊目的のために、クライアントマシン上でローカルにセットを使用したい場合があります。このような場合、クライアントセットを使用します。

例題

```
 ` インタープロセスセット
USE SET("<>Deleted Records")
CREATE SET([Customers];"<>Customer Orders")
If(Records in set("<>Selection"+String($i))>0)
 ` プロセスセット
   USE SET("Deleted Records")
   CREATE SET([Customers];"Customer Orders")
   If(Records in set("Selection"+String($i))>0)
 ` クライアントセット
      USE SET("$Deleted Records")
      CREATE SET([Customers];"$Customer Orders")
      If(Records in set("$Selection"+String($i))>0)
```

命名セレクション

スコープに基づき、2つのタイプの命名セレクションがあります:

- インタープロセス命名セレクション
- プロセス命名セレクション

インタープロセス命名セレクション
インタープロセス命名セレクションの名前は、名前の先頭にインタープロセス (<>) 記号が付きます。インタプロセス命名セレクション名は、インタープロセス (<>) 記号を除いて255文字以内で指定します。

プロセス命名セレクション
プロセス命名セレクションの名前を表す文字列式を使用してプロセスセットを表します (<>記号も$記号も名前の先頭につきません) 。インタプロセスセット名は255文字以内で指定します。

例題

```
 ` インタープロセス命名セレクション
USE NAMED SELECTION([Customers];"<>ByZipcode")
 ` プロセス命名セレクション
USE NAMED SELECTION([Customers];"ByZipcode")
```

プロセス

シングルユーザ版およびクライアント/サーバ版のクライアント側において、2種類のプロセスがあります:

- グローバルプロセス
- ローカルプロセス

グローバルプロセス
文字列 ($記号以外から始まる) を使用してグローバルプロセスの名前を表します。グローバルプロセス名は、255文字以内で指定します。

ローカルプロセス
名前の前にドル記号 ($) をつけてローカルプロセスを表します。ローカルプロセス名は、ドル ($) 記号を除いて255文字以内で指定します。

例題

```
 ` グローバルプロセス"Add Customers"を開始する
$vlProcessID:=New process("P_ADD_CUSTOMERS";48*1024;"Add Customers")
 ` ローカルプロセス"$Follow Mouse Moves"を開始する
$vlProcessID:=New process("P_MOUSE_SNIFFER";16*1024;"$Follow Mouse Moves")
```

命名規則のまとめ

次の表は、4Dの命名規則についてまとめています。

タイプ	最大長	例
テーブル	31	[Invoices]
フィールド	31	[Employees]Last Name
インタープロセス変数	<> + 31	<>vlNextProcessID
プロセス変数	31	vsCurrentName
ローカル変数	$ + 31	$vlLocalCounter
フォーム	31	"My Custom Web Input"
フォームオブジェクト	31	"MyButton"
インタープロセス配列	<> + 31	<>apTables
プロセス配列	31	asGender
ローカル配列	$ + 31	$atValues
メソッド	31	M_ADD_CUSTOMERS
プラグインメソッド	31	WR INSERT TEXT
インタープロセスセット	<> + 255	"<>Records to be Archived"
プロセスセット	255	"Current selected records"
クライアントセット	$ + 255	"$Previous Subjects"
命名セレクション	255	"Employees A to Z"
インタープロセス命名セレクション	<> + 255	"<>Employees Z to A"
ローカルプロセス	$ + 255	"$Follow Events"
グローバルプロセス	255	"P_INVOICES_MODULE"
セマフォ	255	"mysemaphore"

名前が重複する場合

特定のオブジェクトが別タイプのオブジェクトと同じ名前を持つ場合 (例えばフィールドがPersonという名前で、変数もPersonという名前の場合) に、4Dはオブジェクトを識別するために優先順位システムを使用します。重複しない名前の使用に関しては開発者自身に委ねられます。

4Dは、メソッドで使用される名前を次の順位で識別します:

1. フィールド
2. コマンド
3. メソッド
4. プラグインメソッド
5. 定義済み定数
6. 変数

例えば4DにはDateという組み込み関数があります。メソッドにDateという名前を付けても、4Dは組み込み関数のDateとして認識し、メソッドとしては認識しません。つまり、そのメソッドの呼び出しはできないということです。しかしフィールドにDateと命名すると、4DはDate関数の代わりにフィールドとして使用します。

制御フロー

メソッドが単純か複雑かに関係なく、開発者は3つのプログラミング構造のうち、1つ以上のそれを常に使用します。プログラミング構造は、メソッド内で文が実行される順序を決定する実行フローをコントロールします。3つのタイプの構造があります:

- シーケンシャル
- 分岐
- ループ

4Dのプログラミング言語には、これらの構造を制御するステートメントがあります。

シーケンシャル構造

シーケンシャル構造は単純な線形構造です。シーケンシャルは、4Dが最初から最後まで次々に実行する一連のステートメントです。例えば:

```
OUTPUT FORM([People];"Listing")
ALL RECORDS([People])
DISPLAY SELECTION([People])
```

オブジェクトメソッドで頻繁に使用される1行から成るルーチンは最も簡単なシーケンシャル構造の例です:

```
[People]Last Name:=Uppercase([People]Last Name)
```

Note: Begin SQL / End SQLキーワードを使用して、シーケンシャル構造が4DのSQLエンジンで実行されるよう区切ることができます。詳細はこれらのキーワードの説明を参照してください。

分岐構造

分岐構造は、条件をテストし、その結果に基づいて異なる流れのパスにメソッドを導きます。条件はTRUEまたはFALSEに評価されるブール式です。まず分岐構造には構造があり、これは2つの選択肢のうちのいずれかにプログラムの流れを導きます。他の分岐構造には構造があり、これは多くの選択肢の中の1つだけにプログラムの流れを導きます。

ループ構造

メソッドの作成にあたって、何度も同じ処理を繰り返すことがあります。これに実現するために、4Dは3つのループ構造を備えています:

-
-
-

ループを制御する方法には、条件が満たされるまでループする方法と、指定した回数だけループする方法の2通りがあります。各ループ構造にはいずれの方法も用いることができますが、While ループとRepeat ループは条件が満たされるまで繰り返す場合に、For ループは指定した回数だけループする場合の利用に適切です。

Note: 4Dはプログラム構造 (If/While/For/Caes of/Repeat) を512レベルの深さ（入れ子）まで記述できます。

If...Else...End if

If...Else...End if による制御フロー構造の正式な構文は以下のようになります。

```
If(Boolean_Expression)
   Statement
Else
   Statement
End if
```

注： Else 部分は、オプションであり省略し、以下のように記述できます。

```
If(Boolean_Expression)
   Statement
End if
```

If...Else...End if構造は、メソッドにテスト（ブール式）がTRUEかFALSEかにより2つの流れを与えます。

ブール式がTRUEの場合は、テストのすぐ後のステートメントを実行し、ブール式がFALSEの場合には、Else 文のすぐ後のステートメントを実行します。

例

```
  ` ユーザーに名前を入力させる
$Find:=Request(Type a name)
If(OK=1)
   QUERY([People];[People]LastName=$Find)
Else
   ALERT("You did not enter a name.")
End if
```

Tips：一方の条件に実行ステートメントがない分岐処理を書けます。

```
If(Boolean_Expression)
Else
   statement
End if
```

または

```
If(Boolean_Expression)
   statement
Else
End if
```

Case of...Else...End case

Case of...Else...End caseによる制御フロー構造の正式な構文は以下のようになります。

```
Case of
   :(Boolean_Expression)
      statement
   :(Boolean_Expression)
      statement
      .
      .
      .

   :(Boolean_Expression)
      statement
   Else
      statement
End case
```

注 : Else 部分は、オプションであり省略し、以下のように記述できます。

```
Case of
   :(Boolean_Expression)
      statement
   :(Boolean_Expression)
      statement
      .
      .
      .

   :(Boolean_Expression)
      statement
End case
```

構造同様に、Case of...Else...End case構造もメソッドに分岐の流れを与えます。
構造との違いは、Case of...Else...End case構造は合理的に、必要なだけのブール式を評価し、ただ一つTRUEとなるステートメントのみを実行することです。

それぞれのブール式の前にはコロン (:) を付けます。この組み合わせ (コロンとブール式) をケースと呼びます。例えば以下の行はケースです。

```
:(bValidate=1)
```

最初にTRUEになったケースの後のステートメントだけを実行します。TRUEになるケースがない場合、ステートメントを何も実行しません (Else 文が指定されていない場合) 。

最後のケースの後にElse 文を含むことができます。すべての選択式がFALSEの場合に、Else 文の後のステートメントを実行します。

例

下記の例は数値変数を判定し、対応する数字をアラートボックスに表示します。

```
Case of
   :(vResult=1)  ` 数値が1の場合
      ALERT("一です")  ` 1の場合のアラートボックス表示
   :(vResult=2)  ` 数値が2の場合
      ALERT("二です")  ` 2の場合のアラートボックス表示
   :(vResult=3)  ` 数値が3の場合
      ALERT("三です")  ` 3の場合のアラートボックス表示
   Else  ` 数値が1,2,3のいずれでもない場合
      ALERT("一、二、三のいずれでもありません")
End case
```

比較するために、同じことを 構文で記述すると以下のようになります。

```
If(vResult=1)  ` 数値が1の場合
   ALERT("一です")  ` 1の場合のアラートボックス表示
Else
   If(vResult=2)  ` 数値が2の場合
      ALERT("二です")  `2の場合のアラートボックス表示
   Else
      If(vResult=3)  ` 数値が3の場合
         ALERT("三です")  ` 3の場合のアラートボックス表示
      Else  ` 数値が1,2,3のいずれでもない場合
         ALERT("一、二、三のいずれでもありません")
      End if
   End if
End if
```

Case of...Else...End case構造の場合には、最初にTRUEになったケースだけを実行します。2つ以上のケースがTRUEの場合は、最初にTRUEになったケースの後のステートメントだけを実行します。

したがって、階層的なテストを実行するときには、階層上で低い位置にある条件がテスト順序ではじめに記述されていることを確認する必要があります。以下の例では、条件1のテスト表記は、条件1と条件2のテスト表記をカバーするので、テスト順序の最後に位置すべきです。次のコードでは最後の条件が検出されることはありません。

```
Case of
   :(vResult=1)
      ... `ステートメント
   :((vResult=1) & (vCondition#2)) `このケースは判定されない
      ... `ステートメント
End case
```

上記の例では、vResult = 1により他の条件を見る前に分岐するので、第2の条件の表記は判定されません。コードが正しく実行されるためには次のように書きます。

```
Case of
   :((vResult=1) & (vCondition#2)) `このケースが最初に判定される
      ... `ステートメント
   :(vResult=1)
      ... `ステートメント
End case
.
```

さらに階層的なテストを実行したい場合、階層コードを使用する必要があります。

Tip : 分岐の中で、ケースに続くステートメントなしで記述することができます。

```
Case of
   :(Boolean_Expression)
   :(Boolean_Expression)
      .
      .
      .
   :(Boolean_Expression)
      statement
   Else
      statement
End case
```

または

```
Case of
   :(Boolean_Expression)
   :(Boolean_Expression)
      statement
      .
      .
      .

   :(Boolean_Expression)
      statement
   Else
End case
```

または

```
Case of
   Else
      statement
End case
```

While...End while

While...End whileによる制御フロー構造の正式な構文は以下のようになります。

```
While(Boolean_Expression)
   statement
End while
```

While...End whileループは、ブール式がTRUEである限り、ループ内のステートメントを実行し続けます。ループのはじめにブール式を評価し、ブール式がFALSEの場合にはループを行いません。

一般に、While...End whileループに入る手前で、ブール式で判定する値を初期化しておきます。通常はブール式がTRUEになるように設定してからループに入ります。

ブール式はループの中で設定されなければなりません。そうでなければ、ループは永久に続くでしょう。
以下の例では、NeverStopがいつもTRUEであるので、ループは永久に続きます。

```
NeverStop:=True
While(NeverStop)
End while
```

もし、メソッドの実行が制御不能になった場合は、トレース機能を使用し、ループを止め、問題点を追跡することができます。メソッドのトレース方法については、の章を見てください。

例

```
CONFIRM("新規にレコード追加しますか？") ` 利用者に新規レコード追加を問い合わせる
While(OK=1) ` 利用者が望む限りループする
   ADD RECORD([aTable]) ` 新規にレコードを追加する
End while ` ループはいつもEnd_whileによって終わります
```

この例では、ループに入る前にCONFIRM コマンドによりシステム変数OKがセットされます。ユーザがダイアログボックスで「OK」ボタンをクリックすると、システム変数OKに1がセットされ、ループを開始します。それ以外の場合はシステム変数OKに0が設定され、ループをスキップします。ループに入ると、ADD RECORD コマンドはループを続けます。
これは、ユーザがレコードを保存した時点で、システム変数OKに1が設定されるからです。ユーザが最後のレコードを取り消した（保存しない）時点で、システム変数OKに0がセットされ、ループは終了します。

Repeat...Until

Repeat...Untilによる制御フロー構造の正式な構文は以下のようになります。

```
Repeat
   statement
Until(Boolean_Expression)
```

Repeat...Untilループは、ループの後でブール式を判定する以外はWhile...End whileループとまったく同じです。つまり、Repeat...Untilループはループを必ず1回は実行しますが、While...End whileループはブール式が最初にFALSEである場合には、ループを実行しません。

その他のWhile...End whileループとの相違点は、ブール式がTRUEになるまでループを続行することです。

例

以下の例を、While...End whileループの例と比較してください。ブール式を、初期化する必要がない点に注目してください。システム変数OKを初期化するCONFIRM コマンドはありません。

```
Repeat
   ADD RECORD([aTable])
Until(OK=0)
```

For...End for

For...End forによる制御フロー構造の正式な構文は以下のようになります。

```
For(Counter_Variable;Start_Expression;End_Expression{;Increment_Expression})
   statement
End for
```

For...End forループは、カウンタ変数によりループをコントロールします。

- Counter_Variableは、数値変数 (実数、整数、倍長整数) で、Start_Expressionに指定した値で初期化されます。
- ループを実行するたびに、任意の引数Increment_Expressionの値が加算されます。Increment_Expressionを指定しない場合、増分値は1になります。
- カウンタがEnd_Expressionの値を超えた時点で、ループを停止します。

重要:Start_Expression、End_Expression、Increment_Expressionの値は、ループの初めで一度だけ評価されます。
これらの数値が変数で指定されている場合、ループ内でこの変数の値を変更してもループは影響を受けません。

Tips:特別な目的のために、カウンタ変数の値を変更することができます。ループ内でカウンタ変数を変更すると、ループはその影響を受けます。

- 通常、Start_ExpressionはEnd_Expressionより小さい。
- Start_ExpressionとEnd_Expressionが等しい場合、1回だけループが行われる。
- Start_ExpressionがEnd_Expressionより大きい場合、Increment_Expressionに負の値を指定しない限り、ループは行われない。

基本的な使用例

1. 以下の例は、100回の繰り返しを行います。

```
For(vCounter;1;100)
  ` 何らかの処理
End for
```

2. 以下の例は、配列anArrayの全ての要素に対して処理を行います。

```
For($vlElem;1;Size of array(anArray))
  ` 配列要素に対する何らかの処理
   anArray{$vlElem}:=...
End for
```

3. 以下の例は、テキスト変数vtSomeTextの全ての文字について処理を行います。

```
For($vlChar;1;Length(vtSomeText))
  ` 一文字づつの何らかの処理 (例えばタブを処理する等)
   If(Character code(vtSomeText[[$vlChar]])=Tab)
  ` ...
   End if
End for
```

4. 以下の例は、テーブル[aTable]のカレントセクションの各レコードについて処理を行います。

```
FIRST RECORD([aTable])
For($vlRecord;1;Records in selection([aTable]))
  ` レコードについての何らかの処理
   SEND RECORD([aTable])
  ` ...
  ` 次レコードへ移動
   NEXT RECORD([aTable])
End for
```

データベースで作成する大部分のFor...End forループは、上記例題のいずれかの形式になるでしょう。

カウンタ変数の減算

ループに際してカウンタ変数を増加させるのではなく、減少させたい場合があります。その場合、Start_ExpressionにEnd_Expressionより大きい値を設定し、Increment_Expressionに負の数を指定する必要があります。次に挙げる例題は、前述の例と同じ処理を逆の順序で行います。

5. 以下の例は、100回の繰り返しを行います。

```
For(vCounter;100;1;-1)
  ` 何らかの処理
End for
```

6. 以下の例は、配列"anArray"の全ての要素に対して処理を行います。

```
For($vlElem;Size of array(anArray);1;-1)
  ` 配列要素に対する何らかの処理
   anArray{$vlElem}:=...
End for
```

7. 以下の例は、テキスト変数"vtSomeText"の全ての文字について処理を行います。

```
For($vlChar;Length(vtSomeText);1;-1)
  ` 一文字づつの何らかの処理 (例えばタブを処理する等)
   If(Character code(vtSomeText[[$vlChar]])=Tab)
  ` ...
   End if
End for
```

8. 以下の例は、テーブル[aTable]のカレントセクションの各レコードについて処理を行います。

```
LAST RECORD([aTable])
For($vlRecord;Records in selection([aTable]);1;-1)
  ` レコードについての何らかの処理
   SEND RECORD([aTable])
  ` ...
  ` 前レコードへ移動
   PREVIOUS RECORD([aTable])
End for
```

カウンタ変数を1より大きな値で増加させる

必要に応じて、Increment_Expression (正または負の値) に、その絶対値が1より大きな値を指定できます。

9. 以下の例は、配列anArrayの偶数要素について処理を行います。

```
For($vlElem;2;Size of array(anArray);2)
  ` 何らかの処理を要素 #2,#4...の偶数要素に対して行う
   anArray{$vlElem}:=...
End for
```

カウンタ変数を変更してループから抜ける

一定回数のループを行いたいが、他の条件が真になった場合はループから抜けたい場合があります。この場合、ループ内で条件判定を行い、判定結果が真であれば、カウンタ変数に終了値を越える値を明示的に設定することで可能となります。

10. 以下の例は、実際に処理が終了するか、あるいは最初にFALSEに設定されているインタープロセス変数<>vbWeStopがTRUEになるまでレコードセレクションを参照します。この変数はON EVENT CALL プロジェクトメソッドで処理されこのメソッドにより処理を中断します。

```
<>vbWeStop:=False
ON EVENT CALL("HANDLE STOP")
  ` HANDLE STOP は<>vbWeStop変数 を Ctrl+ピリオド(Windows) またはCmd+ピリオド(Macintosh)がタイプされた場合にTrueにします
$vlNbRecords:=Records in selection([aTable])
FIRST RECORD([aTable])
For($vlRecord;1;$vlNbRecords)
  ` 何らかのレコード処理
   SEND RECORD([aTable])
  ` ...
  ` 次のレコードに移動
   If(<>vbWeStop)
      $vlRecord:=$vlNbRecords+1 ` カウンタ変数値をループ終了値以上に設定し、ループを抜ける
   Else
      NEXT RECORD([aTable])
   End if
End for
ON EVENT CALL("")
If(<>vbWeStop)
   ALERT("処理は中断されました。")
Else
   ALERT("正常終了しました。")
End if
```

ループ構造を比較する

For...End forループの例をもう一度見てみましょう。

以下の例は、100回の繰り返しを行います。

```
For(vCounter;1;100)
  ` 何らかの処理
End for
```

ループとループで、同じ処理を実行する方法を調べてみましょう。

以下の例は、同じ処理を実行するループです。

```
$i :=1 `カウンタの初期化
While($i<=100) ` 100回のループ
  ` 何らかの処理
   $i :=$i +1 ` カウンタの増加は必須 (自分で書く)
End while
```

同じ事をループで記述すると以下のようになります。:

```
$i :=1 ` カウンタ変数
Repeat
  ` 何らかの処理
   $i :=$i +1 ` カウンタの増加は必須 (自分で書く)
Until($i=100) ` 100回のループ
```

Tip:For...End forループは、やループよりも高速です。これは4Dが内部的にカウンタ変数のテストおよび増加を行うからです。可能な限りFor...End forループの使用を推奨します。

For...End for ループの最適化

カウンタ変数 (インタープロセス、プロセス、ローカル変数) には実数、整数、倍長整数タイプを使用します。数多く繰り返されるループの場合、とくにコンパイルモードでは、倍長整数タイプのローカル変数を使用してください。

11. 以下に例を示します

```
C_LONGINT($vlCounter) ` 倍長整数変数を使用します<gen9>
For($vlCounter;1;10000)
  ` 何らかの処理
End for</gen9>
```

For...End for ネスト (入れ子) 構造

必要に応じて制御構造をネストすることができます。For...End forループも同じです。誤りを避けるため、各ループ構造ごとに別のカウンタ変数を使用してください。

次に例を示します。

12. 以下の例は二次元配列の全要素への処理です。

```
For($vlElem;1;Size of array(anArray))
  ` ...
  ` 行の何らかの処理
  ` ...
   For($vlSubElem;1;Size of array(anArray{$vlElem}))
  ` 何らかの配列要素への処理
      anArray{$vlElem}{$vlSubElem}:=...
   End for
End for
```

13. 以下の例は、データベースのすべての日付フィールドに対するポインタの配列を作成します。

```
ARRAY POINTER($apDateFields;0)
$vlElem:=0
For($vlTable;1;Get last table number)
   If(Is table number valid($vlTable))
      For($vlField;1;Get last field number($vlTable))
         If(Is field number valid($vlTable;$vlField))
            $vpField:=Field($vlTable;$vlField)
            If(Type($vpField->)=Is Date)
               $vlElem:=$vlElem+1
               INSERT IN ARRAY($apDateFields;$vlElem)
               $apDateFields{$vlElem}:=$vpField
            End if
         End if
      End for
   End if
End for
```

メソッド

コマンド、演算子、および言語の他の部分を動作させるためにメソッドがあります。メソッドにはいくつかの種類があり、オブジェクトメソッド、フォームメソッド、テーブルメソッド（トリガ）、プロジェクトメソッド、データベースメソッドがあります。この章では、すべてのタイプのメソッドに共通の機能を説明します。

メソッドは、ステートメントで構成されます。ステートメントとは、メソッドの1行のことで1つの命令を実行します。ステートメントは単純な場合もあれば、複雑な場合もあります。ステートメントは常に1行ですが最大32,000文字まで使用することができます。これは、ほとんどの処理で十分な長さです。

例えば、以下の行は[People]テーブルに新しいレコードを追加するステートメントです。

```
ADD RECORD([People])
```

メソッドは、**テストとループ**の制御フローの実行を含みます。制御フローに関する詳細は、の節を参照してください。

注：メソッドは最大2GBのテキストまたは、32000行まで記述できます。これらの限界を越えた場合、これ以上行を追加できないというアラートボックスが表示されます。

メソッドの種類

4Dのメソッドには、以下の5種類があります。

- **オブジェクトメソッド**：オブジェクトメソッドとは、オブジェクトのプロパティとして記述される短いメソッドのことです。一般に、オブジェクトメソッドはそのフォームが表示または印刷されているときにオブジェクトを管理します。ユーザがオブジェクトメソッドを呼び出すことはありません。オブジェクトメソッドが属しているオブジェクトをイベントが含んでいる場合に、4Dがオブジェクトメソッドを自動的に呼び出します。
- **フォームメソッド**：フォームメソッドとは、フォームのプロパティです。ユーザはフォームメソッドを使用してデータとオブジェクトの管理を実行することができます。ただし、上述の目的には、オブジェクトメソッドを使用する方が通常は簡単であり、より効果的です。ユーザがフォームメソッドを呼び出すことはありません。

フォームメソッドが属しているフォームをイベントが含んでいる場合に、4Dがフォームメソッドを自動的に呼び出します。

オブジェクトメソッドとフォームメソッドの詳細に関しては、Form eventの節や*4D Design Reference*を参照してください。

- **トリガ**：トリガとは、テーブルのプロパティです。ユーザがトリガを呼び出すことはありません。ユーザがテーブル（追加、削除、修正）のレコードを操作する度にトリガは4Dデータベースエンジンによって自動的に呼び出されます。トリガはユーザのデータベースのレコードに対して「不正な」操作が行われることを防ぎます。例えば、「送り状」システムでは、送り状を送付した顧客名を明記せずに第三者が送り状を追加することを防ぎます。トリガはとても強力なツールで、テーブル上の操作の制限を実行できます。

同様に、思いがけないデータの損失や不正な変更も防ぐことができます。ユーザは簡単なトリガを作成し、それを徐々に洗練されたものにしてゆくことができます。

トリガについての詳細は、の章を参照してください。

- **プロジェクトメソッド**：オブジェクトメソッド、フォームメソッド、トリガはすべて特定のオブジェクト、フォーム、テーブルとそれぞれ関連があります。これとは異なり、プロジェクトメソッドはユーザのデータベースを通して使用可能となります。

プロジェクトメソッドは再利用でき、他のメソッドによる使用が可能です。作業の反復が必要になった場合でも、状況ごとに同じメソッドを書く必要はありません。必要に応じていつでもプロジェクトメソッドを他のプロジェクトメソッド、フォームオブジェクト、フォームメソッドから呼び出すことができます。
呼び出されたプロジェクトメソッドは、その処理を終えると、自分を呼び出した元の場所へと制御を戻します。他のメソッドから呼び出されたプロジェクトメソッドはしばしば「サブルーチン」と呼ばれます。結果を返すプロジェクトメソッドは、関数とも呼ばれます。

プロジェクトメソッドには、メニューと関連付けるというもう1つの使用方法があります。開発者がプロジェクトメソッドをメニューに関連付けると、そのメニューが選択されたときにそのメソッドが実行されるようになります。これでメニューをプロジェクトメソッドの呼び出しと同じように考えることができます。

プロジェクトメソッドに関する詳細は、の節を参照してください。

- データベースメソッド：フォーム内でイベントが生じるときにオブジェクトとフォームのメソッドが呼び出されるのと同様に、作業セッションのイベントが生じると呼び出されるデータベースと連結するメソッドがあります。これが、データベースメソッドです。例えば、ユーザがデータベースを開くたびに、作業セッション中に使用する変数をいくつか初期化したいとします。このために、ユーザはを使用します。これはユーザがデータベースを開くときに、4Dによって自動的に実行されます。

データベースメソッド関する詳細は、の節を参照してください。

メソッド例

メソッドは先頭の行から始まり、最後の行に到達するまで、各ステートメント（命令文）を実行します。以下にプロジェクトメソッドの例を示します。

```
QUERY([People]) ` 検索エディタ画面を表示します
If(OK=1) ` OKボタンをユーザーがクリックした
   If(Records in selection([People])=0) ` 該当するレコードがみつからなかったならば…
      ADD RECORD([People]) ` レコードを追加する
   End if
End if ` これで終わり
```

まず、プログラミング言語の用語と機能を説明することにします。上記の各行を**ステートメント**またはコードと呼びます。プログラミング言語を使用して作成したものを、単にコードとも呼びます。4Dは、コードで指定した処理を実行します。

それでは、最初の行を詳しく見てみましょう。

```
QUERY([People]) ` 検索エディタ画面を表示しますr
```

この行の最初の要素であるQUERY は、コマンドです。コマンドは4Dのプログラミング言語の一部で、処理を実行します。QUERY コマンドは「クエリ」エディタを表示します。「ユーザ」モードの「レコード」メニューから「クエリ」→「検索」を選択することと同じ機能です。

この行の2番目の要素である括弧は、QUERY コマンドに対する引数（パラメータ）を指定します。引数は、コマンドが処理を実行するために必要なものです。この例では、[People]はテーブル名です。テーブル名は常に角カッコ（[...]）の中で指定します。つまり、PeopleテーブルがQUERY コマンドの引数であるということを意味します。コマンドの中には複数の引数を持つものもあります。

3番目の要素は、行の終りに指定されたコメントです。コメントは逆アポストロフィ（` ）によって示します。開発者やコードを解析する人にこのコードが何を行っているのかを説明します。またコメント記号に続く内容は、コードを実行する時点で無視されます。
コメントはそれ自体で1つの行になりますが、通常は上記の例で示すようにコードの右側に記述します。コメントを使用することにより、メソッドの内容を読みやすく、理解しやすいものにします。

注：コメントは32000文字まで記述できます。

以下の行は、レコードが見つかったかどうかを調べます。

```
If(Records in selection([People])=0) ` 該当するレコードがみつからなかったならば…
```

Ifステートメントはフロー制御文です。If文はテストを行い、結果がTrueであれば次の行を実行します。Records in selection は関数です。これは値を返すコマンドです。ここでは、Records in selection 関数は引数として渡されたテーブルのカレントセレクションのレコード数を返します。

注：関数名の頭文字だけが大文字になっていることに注意してください。これは、4Dの関数に対する命名規則によるものです。

既に、カレントセレクションとは何かについて説明しました。これは、その時点で作業対象となっているレコードの集まりのことです。レコードの数が0件の場合（レコードが全く見つからない場合）に以下の行を実行します。

```
ADD RECORD([People]) ` レコードを追加する
```

ADD RECORD コマンドは入力フォームを表示し、新しいレコードを追加します。この行はインデントされています。4Dは、自動的にコードをフォーマットします。この行は、先ほどのフロー制御ステートメント（If）に従属することを示すためにインデントされています。

```
End if ` これで終わり
```

End if 文はIf 文の制御セクションを終了します。フロー制御ステートメントを使用した場合は、常に制御が終了する場所を示す対応ステートメントを指定する必要があります。

この節の概念をしっかりと把握し、理解してください。

次は

もっとよく知るには、以下の節を参照してください:

- オブジェクトメソッドとフォームメソッドについてはForm eventの章へ。
- トリガについてはトリガの節へ。
- プロジェクトメソッドについてはプロジェクトメソッドの節へ。
- データベースメソッドについてはデータベースメソッドの節へ。

プロジェクトメソッド

プロジェクトメソッドには、適切な名前を付ける必要があります。フォームメソッドやオブジェクトメソッドはフォームやオブジェクトと密接に関連付けられていますが、プロジェクトメソッドはどこにでも使用できます。データベースの特定のオブジェクトに付属しているわけではありません。プロジェクトメソッドはその実行方法や使用方法に応じて、次のような役割を果たします:

- メニューメソッド
- サブルーチン、関数
- プロセスメソッド
- イベント処理メソッド
- エラー処理メソッド

これらの用語はプロジェクトメソッドを、それがなんであるかで識別するのでなく、何を行うで識別しています。

メニューメソッドは、カスタムメニューから呼び出されるプロジェクトメソッドです。これは、ユーザのアプリケーションの流れを管理します。メニューメソッドは、必要とされる場所での分岐、フォームの表示、レポートの生成、ユーザデータベースの一般的な管理といった制御を行います。

サブルーチンは、処理の下請け的なプロジェクトメソッドです。他のメソッドから呼ばれて、リクエストされた処理を実行します。関数は、呼び出し元のメソッドに値を返すサブルーチンです。

プロセスメソッドは、プロセスの開始時に呼び出されるプロジェクトメソッドです。このプロセスは、プロセスメソッドが実行されている間だけ存続します。プロセスに関する詳細はの節を参照してください。メニューに属するメニューメソッドのプロパティとして**新規プロセス開始**をチェックした場合も、プロセスメソッドとして開始されます。

イベント処理メソッドは、イベントを処理するプロセスメソッドとして、分離されたプロセス内で実行されます。通常、開発者はイベント管理の大部分を4Dに任せます。例えば、データ入力中、4Dがキーストロークやクリックを検出し、それから正しいオブジェクトとフォームメソッドを呼び出します。このため開発者は、これらのメソッド内でイベントに対し適切に応答できるのです。一方、開発者がイベントを直接操作したい場合があります。例えば（レコードをブラウズするループ等）処理時間の長い操作を実行する場合、「Ctrl+.(ピリオド)」キー（Windows）や「command+.(ピリオド)」キー（Macintosh）を押して、その操作への割り込みを行いたいとします。この場合、開発者はイベント処理メソッドを使用する必要があります。詳細はON EVENT CALL コマンドの説明を参照してください。

エラー処理メソッドは、割り込みを実行するプロジェクトメソッドです。エラーや例外が起こる度に、エラー管理メソッドがインストールされたプロセス内で実行されます。詳細はON ERR CALL コマンドの説明を参照してください。

メニューメソッド

アプリケーションモードでカスタムメニューを選択すると、そのメニューに関連付けられたメニューメソッドが実行されます。メニューエディタを使用して、メニューコマンドにメソッドを割り当てます。メニューが選択されると、それに対応するメニューコマンドが実行されます。この手順は、データベースをカスタマイズする主要な方法の一つです。特定の処理を実行するメニューメソッドを割り当てたカスタムメニューを作成することで、データベースをカスタマイズすることができます。詳細は、4D Design Referenceのメニューエディタの説明を参照してください。

カスタムメニューコマンドにより、単一または複数の処理を実行することができます。例えば、データの入力処理を実行するメニューは、以下の2つの処理を実行するメソッドを呼び出すことができます。まず適切な入力フォームを表示します。次にユーザがキャンセルするまでの間ADD RECORD コマンドによるデータ入力を繰り返します。

連続した処理の自動化は、プログラミング言語の強力な機能の1つです。カスタムメニューを使用すると、処理を自動化することができ、データベースのユーザにより多くのガイダンスを提供することができます。

サブルーチン

プロジェクトメソッドを作成すると、それは同じデータベースシステムの言語の一部になります。プロジェクトメソッドは、4Dに組み込まれたコマンドと同様に呼び出すことができます。このように使用されるプロジェクトメソッドをサブルーチンと呼びます。

サブルーチンは、以下のようなメリットがあります。

- 重複したコードをなくす
- メソッドの役割を明確にする
- メソッドの変更を容易にする
- コードをモジュール化する

例えば、顧客データベースがあるとします。データベースをカスタマイズしていくうちに同じ処理を繰り返し行うことに気づいたとします。それは顧客を検索してレコードを修正するという一連の作業です。そのコーディングは以下のようになっています:

```
  ` 顧客を検索
 QUERY BY EXAMPLE([Customers])
  ` 入力フォームを選択
 FORM SET INPUT([Customers];"Data Entry")
  ` 顧客のレコードを修正
 MODIFY RECORD([Customers])
```

サブルーチンを使用しなければ、顧客レコードの修正を実行するたびにコードを作成しなければなりません。10箇所で同じ処理が必要になると、同じコーディングを10回も行わねばなりません。サブルーチンを使用すれば1回コーディングするだけで済みます。これがコーディングの重複をなくすというサブルーチンの第一の利点です。

先ほど説明したコードがMODIFY CUSTOMERと呼ばれるメソッドであるならば、他のメソッドの中でメソッドの名前を使って実行できます。例えば、顧客のレコードを修正し、それからレコードをプリントするために、以下のようなメソッドを書くことができます:

```
 MODIFY CUSTOMER
 PRINT SELECTION([Customers])
```

この機能はメソッドを劇的にに簡素化します。例においてMODIFY CUSTOMERメソッドがどのように動作するか知る必要がなく、何を行うかを知っていればよいのです。これは、メソッドをサブルーチン化にすることにより処理内容が明確になる、二番目のメリットです。また、これにより作成したメソッドは4D言語を拡張したことになります。

このデータベース例で、顧客を検索する方法を変更する必要がある場合、10か所ではなく、たった1つのメソッドを変更するだけで済みます。これがサブルーチンを使うもう一つの理由です。

サブルーチンを使ってコードをモジュール化します。これはコードをモジュール(サブルーチン)に分割することを意味し、それぞれは論理的な仕事を実行します。アカウントデータベースの以下のコードを見てみましょう:

```
 FIND CLEARED CHECKS ` 決済済みの小切手を見つける
 RECONCILE ACCOUNT ` 口座の照合
 PRINT CHECK BOOK REPORT ` 出納記録の印刷
```

データベースを知らない人でも、このプログラムが何をしているかはわかります。各サブルーチンの処理手順を知る必要はありません。各サブルーチンは長く、複雑な処理で構成されていることもありますが、そのサブルーチンが何を実行するのかだけを知っていれば十分です。

プログラムを論理的な処理単位やモジュールにできるだけ分割することをお勧めします。

メソッドに引数を渡す

メソッドにデータを渡す必要がしばしば発生します。これは引数によって容易にできます。

引数は、メソッド内で行う処理に必要なデータです。引数という用語はこのマニュアルの随所で使用されています。引数は、4Dコマンドへデータを渡す場合にも使用します。以下の例は、文字列"Hello"という引数をALERTコマンドへ渡します:

```
 ALERT("Hello")
```

引数は、メソッドに対しても同じように渡すことができます。例えばメソッドDO SOMETHING が3つの引数を受け取る場合、このメソッドを呼び出すには以下のように書きます:

```
 DO SOMETHING(WithThis;AndThat;ThisWay)
```

引数は、セミコロン (;) で区切ります。

サブルーチン (呼び出されるメソッド) 内で、それぞれの引数の値は自動的に、順に番号が付けられたローカル変数 ($1, $2, $3...) に格納されます。ローカル変数の番号は、引数の順序を表わします。

このローカル変数/引数は**呼び出しメソッド**から渡された実際のフィールドや変数、式ではなく、渡された値を保持しているだけです。

サブルーチン内で、他のローカル変数と同様にこれらの引数 ($1, $2...) を使用できます。

Note: しかしながら、引数として渡した変数の値を変更するコマンドを使用する場合 (例えばFind in field)、$1, $2などの直接使用することはできません。まず標準のローカル変数等にコピーする必要があります (例: $myvar:=$1)。

引数はローカル変数であるため、サブルーチン内でのみ使用可能で、サブルーチンの終了時にクリアされます。このためサブルーチンは、引数として渡された実際のフィールドや変数の値を呼び出しメソッドレベルで変更できません。例えば:

```
  ` MY METHODメソッド
 DO SOMETHING([People]Last Name) ` [People]Last Nameの値が"williams"だとします
 ALERT([People]Last Name)

  ` DO SOMETHINGメソッド
 $1:=Uppercase($1)
 ALERT($1)
```

DO SOMETHINGが表示するアラートボックスには"WILLIAMS"と表示され、MY METHODが表示するアラートボックスには"williams"と表示されます。メソッドは引数$1の値をローカルに変更します。しかしこの変更はMY METHODで引数として渡されたフィールド [People]Last Nameの値に影響しません。

メソッドDO SOMETHINGでフィールドの値を変更する方法には2通りあります:

1. メソッドにフィールドを渡すのではなく、フィールドへのポインタを渡します:

```
  ` MY METHODメソッド
 DO SOMETHING(->[People]Last Name) ` [People]Last Nameの値が"williams"だとします
 ALERT([People]Last Name)

  ` DO SOMETHINGメソッド
 $1->:=Uppercase($1->)
 ALERT($1->)
```

ここで、引数はフィールドではなく、フィールドへのポインタです。DO SOMETHINGメソッド内では、$1はフィールドの値ではなく、フィールドへのポインタです。$1で**参照**されるオブジェクト (上記のコードでは$1->) は、実際のフィールドです。その結果、参照されたオブジェクトの変更はサブルーチンのスコープを超え、実際にフィールドに影響が及びます。この例題では、両方のアラートボックスに"WILLIAMS"と表示されます。

ポインタに関する詳細は、の節を参照してください。

2. メソッドDO SOMETHINGに処理を行わせるだけではなく、値を返すようにコードを書き換えることができます:

```
  ` MY METHODメソッド
 [People]Last Name:=DO SOMETHING([People]Last Name) ` [People]Last Nameの値が"williams"だとします
 ALERT([People]Last Name)

  ` DO SOMETHINGメソッド
 $0:=Uppercase($1)
 ALERT($0)
```

サブルーチンから値が返される2番目の手法は関数と呼ばれ、次の項で説明します。

上級プログラミング: サブルーチン内の引数にはローカル変数$1, $2...を使用してアクセスできます。さらに引数を省略可能とし、${...}シンタックスを使用して参照することもできます。詳細は Count parameters関数の説明を参照してください。

関数: 値を返すプロジェクトメソッド

メソッドからデータを返すこともできます。値を返すサブルーチンを**関数**と呼びます。

値を返す4Dコマンドや4Dプラグインコマンドも関数と呼びます。

以下の行は、文字列のデータ長を返すLength関数を用いたステートメントです。このステートメントは、Length関数がMyLengthという変数に値を返します。

```
 MyLength:=Length("How did I get here?")
```

どのようなサブルーチンでも値を返すことができます。返す値をローカル変数$0に格納します。

例えばUppercase4という以下の関数は、始めの4文字を大文字に変換した文字列を返します:

```
 $0:=Uppercase(Substring($1;1;4))+Substring($1;5)
```

以下は、Uppercase4 関数を使用するメソッドの例です:

```
 NewPhrase:=Uppercase4("This is good.")
```

変数NewPhrase には"THIS is good."が格納されます。

戻り値 $0はサブルーチン内でローカル変数です。サブルーチンの中では$0を通常のローカル変数と同様に使用できます。例えば、前例のDO SOMETHINGにおいて、$0は最初に大文字に変換した$1の値を割り当てられ、その後ALERT コマンドの引数として使われました。サブルーチンの中では、他のローカル変数と同じ方法で$0を使うことができます。サブルーチンが終わる時の$0の値を呼び出し元のメソッドに戻すのは4Dの役割です。

プロジェクトメソッドの再帰呼び出し

プロジェクトメソッドは、自分自身を呼び出すことができます。例えば:

- メソッドAがメソッドBを呼び出し、メソッドBはメソッドAを呼び出します。
- メソッドAは自身を呼び出すことができます。

これは**再帰呼び出し**と呼ばれています。4D言語は再帰呼び出しを完全にサポートしています。

例題を見てみましょう。以下のフィールドから成る[Friends and Relatives] テーブルがあります:
- [Friends and Relatives]Name
- [Friends and Relatives]ChildrensName

この例題では、フィールドの値は重複しない、すなわち同じ名前の人間はいないとします。名前を指定することで、以下のような文を作成します:"A friend of mine, John who is the child of Paul who is the child of Jane who is the child of Robert who is the child of Eleanor, does this for a living!":

1. この文を以下のように作成できます:

```
 $vsName:=Request("Enter the name:";"John")
 If(OK=1)
    QUERY([Friends and Relatives];[Friends and Relatives]Name=$vsName)
    If(Records in selection([Friends and Relatives])>0)
       $vtTheWholeStory:="A friend of mine, "+$vsName
       REPEAT
          QUERY([Friends and Relatives];[Friends and Relatives]ChildrensName=$vsName)
          $vlQueryResult:=Records in selection([Friends and Relatives])
          If($vlQueryResult>0)
             $vtTheWholeStory:=$vtTheWholeStory+" who is the child of "+[Friends and Relatives]Name
             $vsName:=[Friends and Relatives]Name
          End if
       Until($vlQueryResult=0)
       $vtTheWholeStory:=$vtTheWholeStory+", does this for a living!"
       ALERT($vtTheWholeStory)
    End if
 End if
```

2. 以下の方法でも作成できます:

```
 $vsName:=Request("Enter the name:";"John")
 If(OK=1)
    QUERY([Friends and Relatives];[Friends and Relatives]Name=$vsName)
    If(Records in selection([Friends and Relatives])>0)
       ALERT("A friend of mine, "+Genealogy of($vsName)+", does this for a living!")
    End if
 End if
```

再帰関数Genealogy ofは以下の通りです:

```
  ` Genealogy of プロジェクトメソッド
  ` Genealogy of ( String ) -> Text
  ` Genealogy of ( Name ) -> Part of sentence

 $0:=$1
 QUERY([Friends and Relatives];[Friends and Relatives]ChildrensName=$1)
 If(Records in selection([Friends and Relatives])>0)
    $0:=$0+" who is the child of "+Genealogy of([Friends and Relatives]Name)
 End if
```

Genealogy ofメソッドが自分自身を呼び出していることに注目してください。

最初に挙げた方法は**反復性のアルゴリズム**です。2番目に挙げた方法は**再帰呼び出しのアルゴリズム**です。

前述の例題のようなコードを実装する場合、反復性や再帰呼び出しを使用してメソッドを書くことができるということに注目してください。一般的に、再帰呼び出しは、より明瞭で、読みやすく、維持しやすいコードを提供します。ただし、この使用は必須ではありません。

4D内での再帰呼び出しの代表的な使用方法は以下の通りです:

- 例題と同じく、お互いに関連するテーブル内でのレコードの取り扱い。
- FOLDER LISTコマンドとDOCUMENT LISTコマンドを使用して、ディスク上にあるドキュメントとフォルダをブラウズする。フォルダにはフォルダとドキュメントが含まれており、サブフォルダはまたフォルダとドキュメントを含むことができます。

重要: 再帰呼び出しは、必ずある時点で終了する必要があります。例えばGenealogy ofメソッドが自身の呼び出しを止めるのは、クエリがレコードを返さないときです。この条件のテストをしないと、メソッドは際限なく自身を呼び出します。(メソッド内で使用される引数やローカル変数と同様に) 再帰呼び出しを行う容量が一杯になると、最終的に4Dは"スタックがいっぱいです"エラーを返します 。

デバッグ

- なぜデバッガを使用するか?
- シンタックスエラーウィンドウ
- デバッガ
- ウォッチエリア
- メソッド連鎖エリア
- カスタムウォッチエリア
- ソースコードエリア
- ブレークポイント
- ブレークリスト
- コマンドのキャッチ
- デバッガのショートカット

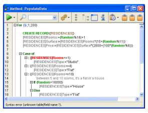
なぜデバッガを使用するか？

メソッドを開発しテストする際には、エラーを発見し修正することが重要です。

言語を使用する場合におこりうるエラーには、タイプミス、シンタックス（構文）または環境エラー、設計やロジックのエラー、ランタイムエラーなどがあります。

タイプミス

タイプミスは**メソッドエディタ**によって検出され、赤色で表示、またメソッドウィンドウ下部の情報エリアにメッセージが表示されます。以下はタイプミス時のウインドウ表示例です:



```
For ($i;1;200)

    CREATE RECORD([RESIDENCES])
    [RESIDENCES]Rooms:=(Random%16)+1
    [RESIDENCES]Surface:=[RESIDENCES]Rooms*(10+(Random%11))
    [RESIDENCES]Price:=[RESIDENCES]Surface*(2800+(100*(Random%8)))

  Case of
    : ([RESIDENSES]Rooms<=1)
        [RESIDENCES]Type:="Studio"
    : ([RESIDENCES]Rooms<=4)
        [RESIDENCES]Type:="Flat"
    : ([RESIDENCES]Rooms<=10)
         `between 5 and 10 rooms, it's a flat or a house.
      If (Random>16000)
          [RESIDENCES]Type:="House"
      Else
          [RESIDENCES]Type:="Flat"
```

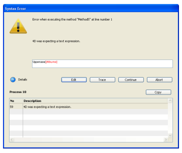
Syntax error (unknown table/field name ?).

このようなタイプミスは通常シンタックスエラーの原因となります (この場合、テーブル名が間違っています)。コードを検証すると、情報エリアにエラーの説明が表示されます。

このような場合タイプミスを修正し、テンキーのenterキーを押すと、再度コードの検証が行われます。メソッドエディタに関する詳細は4D Design Referenceを参照してください。

シンタックスまたは環境エラー

メソッドの実行時のみとらえることのできるエラーがあります。シンタックスエラーウィンドウはエラーが発生した際に表示されます:



Error when executing the method "Method5" at line number 1

4D was expecting a text expression.

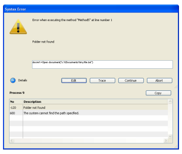
Uppercase([Albums])

Process 10

No	Description
59	4D was expecting a text expression.

このウィンドウでは、文字式を受け入れるUppercaseコマンドにテーブル名が渡されるというエラーが表示されています。このウィンドウやボタンの利用方法はシンタックスエラーウィンドウを参照してください。上の画像では、"詳細"エリアが展開され、最新のエラーと番号が表示されています。

時に、配列やBLOBを作成するための十分なメモリがない場合があります。ディスク上のドキュメントにアクセスしようとしたときに、ドキュメントが存在しないか、他のアプリケーションにより既に開かれていることもあります。



Error when executing the method "Method5" at line number 1

Folder not found

docref:=Open document("c:\\Documents\\myfile.txt")

Process 9

No	Description
-120	Folder not found
600	The system cannot find the path specified.

このようなエラーはコードやその書き方を直接の原因として発生するわけではありません。これらは時に"エラーの原因となるものがたまたま起こった"ために発生します。ほとんどの場合、ほとんどの場合、このようなエラーはON ERR CALLコマンドでインストールされるエラー処理メソッドで簡単に処理できます (ON ERR CALLの説明参照)。

このウィンドウに関する詳細はシンタックスエラーウィンドウを参照してください。

設計またはロジックエラー

一般にこれらは発見が最も難しいタイプのエラーです。を使用して、それらを検知します。これまでに説明しているエラーは、タイプミスを除いて、「設計またはロジックのエラー」という範疇に該当します。例は次の通りです:

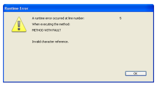
- まだ初期化されていない変数を用いようとしたため、シンタックスエラーが発生する場合があります。
- 間違った引数を受け取ったサブルーチンが、その間違った名前によりドキュメントを開こうとしたため、環境エラーが発生している場合があります。この場合、実際に中断が発生しているコード部分が問題の原因ではなく、外部にあるということに注意が必要です。

設計またはロジックのエラーには、次のような場合もあります:

- SAVE RECORDコマンドを呼び出す際に、対象となるレコードがロックされているかどうかを最初にテストしなかったために、レコードが正しく更新されない。
- オプション引数を追加した状態がテストされていないため、メソッドが想定通りに動作しない。

ランタイムエラー

アプリケーションモードでは、インタプリタモードでは決して見られない次のようなエラーが発生する場合があります:



A runtime error occurred at line number: 5

When executing the method:

METHOD WITH FAULT

Invalid character reference.

これは、文字列の長さを超える文字の場所を参照しようとしているということを示しています。問題の原因を迅速に発見するには、メソッドの名前と行番号を記録し、ストラクチャファイルのインタプリタ版を再び開いて、メソッドの指定された行を確認します。

エラーが発生した場合の対応

エラーは日常的なものです。相当行数（数百行程度）のプログラム・コードをエラーが発生しないように作成できることは、非常にまれです。むしろ、エラーに対応または修正するほうが普通です。

4Dはマルチタスク対応アプリケーションなので、ウインドウを切り替えるだけで、すばやくメソッドを編集し、実行することができます。そのたびメソッド全体を再実行する必要がないため、失敗やエラーを非常に迅速に修正できます。また、を使用すると、エラーを迅速に検出できます。

エラー検出の際によくある初歩的な失敗は、シンタックスエラーウィンドウのアボートボタンをクリックし、メソッドエディタに戻り、コードを表示して原因を確認しようとすることです。これは絶対に止めてください。**常に**を使用すれば、相当の時間と労力を節減することができます。

・予期しないシンタックスエラーが発生したときは、を使用します。
・環境エラーが発生した場合には、を使用します。
・その他どのようなタイプのエラーが発生した場合でも、を使用します。

ほとんどの場合、は、エラーが発生した理由を知るために必要な情報を表示します。この情報があれば、エラーの修正方法はわかります。

Tip: の使用法を数時間費やして学習し、実際に試しておけば、エラーの原因を究明しなければならなくなった時に何日分、何週間分もの時間と労力をかけずにすむことなります。

を使用するもう1つの理由は、コードの作成です。いつも以上に複雑なアルゴリズムを作成してしまう場合があります。達成感こそありますが、コーディングが正しいかどうか、テストする前でもまったく確かではありません。見当もつかないまま実行するのではなく、コードの最初でTRACEコマンドを使用します。その後、コードをステップごとに実行して動作を制御し、不安が的中するかどうかを確認します。完全主義的な考え方では、このような方法は望ましくないかもしれませんが、現実主義的な方法が報われる場合もあります。いずれにしても、を使用してください。

結論

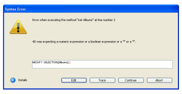
デバッガを使用しましょう。

シンタックスエラーウィンドウ

メソッドの実行が停止されると**シンタックスエラーウィンドウ**が表示されます。メソッドの実行は主に以下の理由で停止されます:

- 以降のメソッド実行を妨げるエラーが発生したため、4Dが実行を停止する。
- メソッドがFalseの表明を生成する (ASSERTコマンド参照)。

シンタックスエラーウィンドウは以下のようなものです:

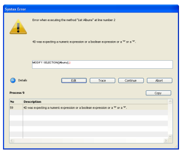

Error when executing the method "List Albums" at line number 2

4D was expecting a numeric expression or a boolean expression or a "*" or a "*".

```
MODIFY SELECTION([Albums];)
```

シンタックスエラーウィンドウの上部テキストエリアには、エラーの説明メッセージが表示されます。下部のテキストエリアにはエラーが発生していた時の行が表示されます。エラーが発生したエリアはハイライトされます。

詳細ボタンをクリックすると、プロセスのエラースタックを表示するウィンドウを展開できます:


Error when executing the method "List Albums" at line number 2

4D was expecting a numeric expression or a boolean expression or a "*" or a "*".

```
MODIFY SELECTION([Albums];)
```

Process 9

No	Description
59	4D was expecting a numeric expression or a boolean expression or a "*" or a "*".

ウィンドウの一番下の部分には**アボート**、**トレース**、**続行**、**編集**、そして (ウィンドウが展開されていれば) **コピー**の5つのボタンがあります。

- **アボート**: メソッドは中断され、メソッドの実行を開始したときの状態に戻ります。イベントに対してフォームメソッドまたはオブジェクトメソッドが実行されている場合 には、いずれの場合にも停止され、フォームに戻ります。メソッドがアプリケーションモードから実行されている場合には、このモードに戻ります。

- **トレース**: トレース/デバッガモードに入り、デバッガ ウィンドウが表示されます。現在行の一部が実行されている場合には、トレースボタンを数回クリックする必要があるかもしれません。 行の実行が終了すれば、デバッガ ウインドウが表示されます。

- **続行**: 実行は継続されます。エラーが発生した行は、その位置によっては一部実行される場合があります。慎重に実行を継続してください。エラーが原因で、メソッド が正常に実行できない場合があります。通常は、継続しようとは思わないでしょう。SET WINDOW TITLE のように、コードの残りの部分の実行やテストの妨げにならない単純な呼び出しでエラーが発生している場合に、続行ボタンをクリックし ても構いません。このようにすれば、より重要なコーディングに集中し、些細なエラーは後で修正することができます。
  Note: Altキー(Windows)または **Option**キー(Mac OS) を押すと、続けるボタンが無視へと変化します。無視をクリックすると、同じメソッドの同じ行でトリガーされた同じエラーが再度起きても、ウィンドウは表示されないことを意味します。このショートカットは、例えばループ中などで繰り返し発生するエラーの場合に便利です。この場合、ユーザーが**続行** ボタンをそのたびにクリックしたかのように、すべてが続行します。

- **編集**: すべてのメソッドの実行は中断されます。4Dはデザインモードに切り替わります。エラーが発生したメソッドがメソッドエディタで表示され、エラーを修正す ることができます。このオプションは、エラーの内容がすぐ判明し、これ以上調査しなくても修正できる場合に使用します。

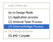
- **コピー**: このボタンをクリックすると、デバッグ情報がクリップボードにコピーされます。この情報はエラーの内部環境 (番号や内部コンポーネント等) を説明します。情報はタブ区切り形式で記述されます。ボタンをクリックしたら、クリップボードの内容を、テキストファイルやスプレッドシート、電子メール などにペーストし、検証できます。

デバッガ

デバッガーという用語は、バグという用語に由来しています。メソッド内のバグとは、除去すべき間違いのことです。エラーが発生した場合、またはメソッドの実行を監視する必要がある場合には、デバッガーを使用します。デバッガでは、メソッドをステップごとにゆっくり確認してメソッドの情報を検証できるため、バグを発見するために役立ちます。このようにメソッドをステップごとに確認する処理は**トレース**と呼ばれます。

次のような方法を使用して、デバッガーウィンドウでメソッドを表示し、トレースできます:

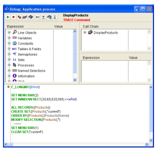
- シンタックスエラーウィンドウで**トレース**ボタンをクリックする。
- TRACEコマンドを使用する。
- メソッド実行ウィンドウで**デバッグ**ボタンをクリックする。
- メソッド実行中にAlt+Shift+右クリック (Windows) または Control+Option+Command+クリック (Macintosh) を行い、表示されるポップアップウィンドウ内でトレースするプロセスを選択する:


User Interrupt
Go to Design Mode
(1) Application process
(2) Internal Timer Process
(3) Internal Bridge Process
(4) Design process
(5) $4D Compiler

- **ランタイムエクスプローラー**の**プロセス**ページにてプロセスを選択した後、**トレース**ボタンをクリックする。
- メソッドエディターウィンドウ、またはランタイムエクスプローラーのブレークおよびキャッチページでブレークポイントを作成する。

注: データベースのパスワードシステムが有効な場合、デザインモードへのアクセス権を持つグループに所属するDesignerやユーザーだけが、メソッドをトレースできます。

次に示すのはデバッガーウィンドウです:


4D Debug: Application process
DisplayProducts
TRACE Command
Expression
Value
Line Objects
Variables
Constants
Tables & Fields
Semaphores
Sets
Processes
Named Selections
Information
Web
Call Chain
DisplayProducts
Expression
Value

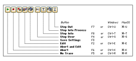
```
C_LONGINT($Wref)

SET MENU BAR(2)
SET WINDOW RECT(20;60;820;580;<>wRef)

ALL RECORDS([Products])
CREATE SET([Products];"current")
ORDER BY([Products];[Products]Name)
MODIFY SELECTION([Products];*)
  `<==>
SET MENU BAR(1)
CLEAR SET("current")
```

必要に応じてデバッガーウィンドウを移動したり、その内部にあるウィンドウ枠のサイズを変更したりできます。新しいデバッガーウィンドウの表示には、同じセッション内で表示された最後のデバッガーウィンドウと同じ構成(ウィンドウのサイズと位置、分割線の配置およびカスタム・ウォッチエリアの内容)を使用します。

4Dはマルチタスク環境です。複数のユーザプロセスを実行した場合には、それぞれのプロセスを個別にトレースできます。プロセスそれぞれについて1つのデバッガーウィンドウを表示できます。

実行制御ツールバーボタン

デバッガーウィンドウの上部にある実行制御ツールバーには、9個のボタンがあります:


Button
Windows
MacOS
Step Out F7 or Ctrl-U ⌘-U
Step Into Process
Step Into F8 or Ctrl-T ⌘-T
Step Over F4 or Ctrl-S ⌘-S
Save Settings F3
Edit F2 or Ctrl-E ⌘-E
Abort and Edit
Abort F6 or Ctrl-K ⌘-K
No Trace F5 or Ctrl-R ⌘-R

トレースなしボタン

トレースが停止され、通常のメソッド実行が再開されます。

Note: **Shift+F5**または**Shift**を押しながらトレースなしボタンをクリックすると、実行が再開されます。この操作により、以降のカレントプロセスでの全てのTRACE呼び出しが無効になります。

アボートボタン

メソッドは中断され、メソッドの実行を開始する前の状態に戻ります。イベントに対して実行しているフォームメソッドまたはオブジェクトメソッドをトレースしている場合には、いずれの場合にも停止され、フォームに戻ります。アプリケーションモードから実行しているメソッドをトレースしていた場合には、そのモードに戻ります。

アボート&編集ボタン

アボートボタンががクリックされた時と同様、メソッドは中断されます。さらに、4Dはメソッドエディタウィンドウを開いて、**アボート&編集** ボタンがクリックされた時点で実行していたメソッドを表示します。

Tip: このボタンは、コードにどのような変更が必要かが明らかであり、メソッドのテストを続行するためにその変更が必要な場合にクリックしてください。変更が完了したら、メソッドを再実行できます。

編集ボタン

編集ボタンをクリックすると、**アボート&編集** ボタンをクリックした場合と同じ動作が実行されますが、現在の実行をアボートしません。メソッドの実行はその時点で一時停止されます。4Dは**編集** ボタンがクリックされた時点で実行されていたメソッドをメソッドエディタウィンドウで表示します。

重要: このメソッドを修正することはできますが、デバッガウィンドウで現在トレースされているメソッドのインスタンスに対しては、この修正内容は反映されず、実行もされません。メソッドがアボートされるか、または正常に終了した後、このメソッドが次に実行される時に修正が反映されます。つまり、メソッドへの変更を有効にするには、メソッドを再ロードしなければなりません。

Tip: このボタンは、コードに必要な変更内容がわかっている場合や、変更が実行やトレースの対象となるコードの残りの部分の妨げにならない場合に使用します。

Tip: オブジェクトメソッドは各イベント毎に再ロードされます。オブジェクトメソッドをトレースしている場合、フォームを終了する必要はありません。オブジェクトメソッドを編集し、変更内容を保存し、フォームに戻って再実行することができます。フォームメソッドのトレースや変更の際に、フォームメソッドを再ロードするには、フォームを終了し、再び表示しなければなりません。フォームを大規模にデバッグする場合のコツは、(デバッグの対象となっている)コードを、フォームメソッドからのサブルーチンとして使用しているプロジェクトメソッドに入力することです。このようにすれば、このサブルーチンがフォームメソッドから呼び出されるたびに再ロードされるため、フォームをトレースし、編集し、再テストしている間もフォームを使用することができます。

設定保存ボタン

現在のデバッガウィンドウの構成(ウィンドウのサイズと位置、分割線の配置およびカスタム・ウォッチエリアの内容)を、データベースが開かれるたびにデフォルトで使用できるように保存することができます。これらの内容は、データベースのストラクチャファイルに保存されます。

ステップ(同一メソッドのみ)ボタン

現在のメソッド行(**プログラムカウンタ**と呼ばれる黄色い矢印で示されている行)が実行され、デバッガは次の行に移動します。**ステップ(同一メソッドのみ)**ボタンはサブルーチンや関数に移動することはなく、現在トレースの対象となっているメソッドのレベルにとどまります。サブルーチンや関数呼び出しもトレースしたい場合には、**ステップ(呼び出しメソッドもトレース)**ボタンを使用します。

ステップ(呼び出しメソッドもトレース)ボタン

別のメソッド(サブルーチンまたは関数)を呼び出す行が実行される場合にこのボタンを使用すると、呼び出されているメソッドがデバッガウィンドウに表示され、このメソッドをステップごとに実行できます。デバッガウィンドウのでは、新しく呼び出されたメソッドがカレント(一番上)となります。別のメソッドを呼び出していない行が実行される場合には、このボタンは**ステップ(同一メソッドのみ)**ボタンと同等に動作します。

ステップ(別プロセスもトレース)ボタン

新しいプロセスを作成する行(New processコマンドが記述されている行)が実行される場合にこのボタンを使用すると、新しいデバッガウィンドウが表示され、新しく作成されたプロセスのプロセスメソッドをトレースすることができます。新しいプロセスを作成しない行が実行される時には、このボタンは**ステップ(同一メソッドのみ)**ボタンと同等に動作します。

呼び出し元へ戻るボタン

サブルーチンや関数をトレースする場合にこのボタンをクリックすると、現在トレースされているメソッド全体を実行し、呼び出し元メソッドに戻ることができます。デバッガウィンドウはメソッド連鎖の前のメソッドに戻ります。現在のメソッドがメソッド連鎖の最後のメソッドである場合には、デバッガウィンドウが閉じられます。

実行制御ツールバーについて

実行制御ツールバーの右側には、デバッガから次のような情報が表示されます:

- 現在トレースしているメソッドの名前(黒で表示されます)
- デバッガウィンドウが表示される原因となった問題(赤で表示されます)

先に示したウィンドウの例では、次の情報が表示されています:

- 現在トレースされているメソッドはDE_DebugDemoメソッドです。
- デバッガウィンドウが表示されているのは、C_DATEコマンドへの呼び出しが検出され、このコマンドは検出の対象コマンドの1つであるためです。

デバッガとメッセージが表示される理由は次の通りです(赤で表示されます):

- **TRACEコマンド**: TRACEの呼び出しが実行されたため。
- **ブレークポイントに到達**: ブレークポイントが発見されたため。
- **ユーザによる割り込み**: Alt+Shift+右クリック(Windows)またはcontrol+option+command+クリック(Macintosh)を使用するか、あるいはデザインモードのランタイムエクスプローラの**プロセス**ページで**トレース**ボタンをクリックしたため。
- **次のコールを検出**: **コマンド名** : 検出の対象となった4Dコマンドへの呼び出しが実行位置にあるため。
- **新規プロセスへステップ中**: **ステップ(別プロセスもトレース)**ボタンを使用したため、新しく作成されたプロセス用に開かれたデバッガウィンドウのためこのメッセージが表示される。

デバッガウィンドウのペイン

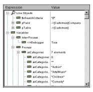
デバッガウィンドウは、前述の実行コントロールツールバーとサイズ変更可能な次の4つのペインから構成されます:

- ウォッチエリア
- メソッド連鎖エリア
- カスタムウォッチエリア
- ソースコードエリア

最初の3つのエリアでは、操作が簡単な階層型リストを使用して、関連するデバッグ情報を表示します。4番目のソースコードエリアは、トレースされているメソッドのソースコードを表示します。それぞれのエリアには、デバッグ作業を支援する独自の機能があります。マウスを使用して、デバッガウィンドウだけではなく、各エリアも垂直および水平方向にサイズを変更することができます。さらに、最初の3つのエリアには、表示する2つのカラムの間に点線による区切り線が含まれています。マウスを使用して、この点線を移動し、必要に応じて水平方向にカラムのサイズを変更することができます。

ウォッチエリア

ウォッチエリアは、デバッグウィンドウの左上隅の実行コントロールツールバーの下に表示されます。以下は表示例です:



Expression	Value
Line Objects	
$vSearchCriteria	"Z"
pField	->[Customers]Company
pTable	->[Customers]
Variables	
Inter-Process	
<>fDebugger	True
Process	
asCategories	7 elements
asCategories	0
asCategorie...	""
asCategorie...	"Action"
asCategorie...	"Arts/Music"
asCategorie...	"Children"
asCategorie...	"Comedy"

ウォッチエリアには、システム、4D環境、および実行環境について役立つ一般情報が表示されます。

式欄には、オブジェクトや式の名前が表示されます。**値**欄には、オブジェクトや式に対応する現在の値が表示されます。

エリア右側にある値をクリックすると、そのオブジェクトの値を変更できる場合には、オブジェクトの値を修正できます。

複数レベルを対象とする階層リストは、メインレベルでテーマごとにまとめられています。テーマは、次の通りです:

- ラインオブジェクト
- 変数
- 定数
- フィールド
- セマフォ
- セット
- プロセス
- 命名セレクション
- インフォメーション
- キャッシュの統計

テーマによっては、各項目に1つまたは複数のサブレベルがある場合もあります。テーマ名の隣にあるリストノード(アイコン)をクリックすると、テーマが拡大、または縮小します。テーマが拡大されている場合には、そのテーマにある項目は見えています。
テーマに複数レベルの情報がある場合には、各項目の隣にあるリストノードをクリックすると、そのテーマで提供されているすべての情報を調べることができます。

どの時点でも、テーマ、テーマサブリスト(あれば)、テーマ項目をカスタムウォッチエリアにドラッグ＆ドロップすることができます。

ラインオブジェクト

このテーマには、次のようなオブジェクトや式の値が表示されます

- 実行されるコードの行(プログラムカウンターにより、ソースコードエリア内で黄色の矢印でマークされている行)で使用されている。
- コードの前の行で使用されている。

コードの前の行とは実行直後の行であるため、ラインオブジェクトテーマでは、その行が実行される前または後の現在の行のオブジェクトや式が表示されます。例えば、次のメソッドを実行した場合を想定します:

```
TRACE
a:=1
b:=a+1
c:=a+b
 // ...
```

1. デバッグウィンドウでソースコードエリアのプログラムカウンターをa:=1の行にセットします。この時点ではラインオブジェクトテーマには、次のように表示されています:

a: **未定義**

変数 a が表示されているのは、実行の対象となっている(ただし、まだ初期化されていない)行で使用されているためです。

2. 1行先にステップすると、プログラムカウンタはb:=a+1の行に設定されます。この時点ではラインオブジェクトテーマには、次のように表示されています:

a: 1
b: **未定義**

変数 a が表示されているのは、ちょうど実行され、数値1を割り当てられたばかりの行で使用されているためです。また、変数 a が表示されているのは、変数 b への割り当て式として実行される行でも使用されているためでもあります。変数 b が表示されているのは、実行の対象となっている(ただし、まだ初期化されていない)行で使用されているためです。

3. 再び1行先にステップすると、プログラムカウンタはc:=a+bの行に設定されます。この時点ではラインオブジェクトテーマには、次のように表示されています:

c: **未定義**
a: 1
b: 2

変数 c が表示されているのは、実行の対象となっている(ただし、まだ初期化されていない)行で使用されているためです。変数 a と b が表示されているのは、1つ前の行で使用され、この行で実行されているからです。

ラインオブジェクトテーマは、とても便利なツールです。ある行が実行される度にカスタムウォッチエリアに式を入力することなく、ラインオブジェクトテーマによって表示される値を検証することができます。

変数

このテーマは、次のサブテーマから構成されます:

- **インタープロセス変数**:この時点で使用されているインタープロセス変数のリストを表示します。インタープロセス変数を使用していない場合には、このリストは空白の場合があります。インタープロセス変数の値は、変更することができます。
- **プロセス変数**:カレントプロセスで使用されている変数のリストを表示します。このリストが空白であることはほとんどありません。プロセス変数の値は、修正することができます。
- **ローカル変数**: 現在トレースの対象となっているメソッド(ソースコードエリアに表示されているメソッド)で使用されているローカル変数のリストを表示します。ローカル変数を使用していない場合や、ローカル変数がまだ作成されていない場合には、このリストは空白の場合があります。ローカル変数の値は、修正することができます。
- **パラメータ(引数)**:メソッドが受け取ったパラメータのリストを表示します。現在トレースの対象となっているメソッド(ソースコードエリアに表示されているメソッド)にパラメータが渡されていない場合には、このリストは空白の場合があります。パラメータの値は、修正することができます。
- **セルフポインタ**:オブジェクトメソッドをトレースしている場合には、現在のオブジェクトへのポインタを表示します。この値を修正することはできません。

Note: 文字列変数、テキスト変数、数値変数、日付変数、および時間変数は、修正することができます。つまり、キーボードを使用して値を入力できる変数は、修正することができます。

他の変数と同様に、配列は、その設定範囲によって、インタープロセス配列サブテーマ、プロセス配列サブテーマ、およびローカル配列サブテーマで表示されます。デバッガは各配列に階層レベルをつけて表示します。このため、配列要素の値がある場合にはこれを取得、または変更することができます。デバッガ は要素ゼロを含む最初の100個の要素を表示します。値欄には、配列名ごとのサイズが表示されます。配列を作成した後、最初のサブ項目には、現在選択されている要素番号、次に要素ゼロ、その次に他の要素(100個まで)が表示されます。文字列配列、テキスト配列、数値配列、および日付配列は、修正することができます。選択された要素番号、要素ゼロ、および他の要素(100個まで)も、修正することができます。配列のサイズを修正することはできません。

Reminder: 個別の配列要素も含め、項目はいつでもウォッチエリアからカスタムウォッチエリアへドラッグ＆ドロップすることができます。

定数

エクスプローラウィンドウの定数ページのように、4Dが提供する定義済み定数を表示します。このテーマの表現式を修正することはできません。

テーブルとフィールド

データベース内にあるテーブルやフィールドを一覧表示しますが、サブフィールドは表示しません。各テーブル項目では、値欄にカレントプロセスのカレントセクションのサイズは勿論、ロックされたレコードナンバー(テーブルアイテムは拡張される)も表示されます。各フィールド項目では、カレントレコードがある場合には、値欄にカレントレコードのフィールドの値(ピクチャ、サブテーブル、BLOBは除く)が表示されます。このテーマでは、フィールドの値を修正することはできますが(ただし、取り消しはできません)、テーブル情報を修正することはできません。

セマフォ

現在設定されているローカルセマフォを一覧表示します。各セマフォでは、値欄にセマフォを設定したプロセスの名前が表示されます。セマフォを使用していない場合、このリストは空白です。このテーマの表現式を修正することはできません。グローバルセマフォは表示されません。

セット

カレント(現在トレースの対象となっている)プロセスで定義されているセットを一覧表示します。インタープロセスセットも一覧表示します。各セットでは、値欄にレコードの数とテーブル名が表示されます。セットを使用していない場合には、このリストは空白の場合があります。このテーマからの表現式を修正することはできません。

プロセス

作業セッションを開始してから起動されたプロセスを一覧表示します。値欄に、それぞれのプロセスの現在の状態(実行中、一時停止等)および使用した時間が表示されます。このテーマからの表現式を修正することはできません。

命名セレクション

カレント(現在トレースの対象となっている)プロセスで定義されているプロセス命名セレクションを一覧表示します。また、インタープロセス命名セレクションも一覧表示します。各命名セレクションでは、値欄にレコード数およびテーブル名が表示されます。命名セレクションを使用していない場合には、このリストは空白の場合があります。このテーマからの表現式を修正することはできません。

インフォメーション

現在のデフォルトテーブル(あれば)等、一般的な情報を表示します。このテーマからの表現式を修正することはできません。

キャッシュの統計

4Dのキャッシュ内にロードされているテーブル、インデックスページおよび命名セレクションの利用に関しての統計を表示します。

コンテキストメニュー

ウォッチエリアのコンテキストメニューで追加オプションを提供します。このメニューを表示するには:

- Windowsでは、マウスの右ボタンを使用してウォッチエリア内の任意の位置をクリックする。
- Macintoshでは、ウォッチエリアの任意の位置でcontrol+クリックを実行する。

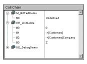
ウォッチエリアのコンテキストメニューが、次のように表示されます:

- **すべて縮める**:ウォッチエリアの階層リストのすべてのレベルを縮小します。
- **すべて拡げる**:ウォッチエリアの階層リストのすべてのレベルを拡張します。
- **タイプ表示**:それぞれのオブジェクトのオブジェクトタイプを(適切な場合に)表示します。
- **フィールド＆テーブル番号表示**:フィールドのそれぞれのテーブルまたはフィールドの番号を表示します。テーブル番号やフィールド番号を用いて作業している場合、またはTableやFieldを使用し、ポインタを用いて作業している場合、このオプションは非常に便利です。
- **アイコン表示**:それぞれのオブジェクトのオブジェクトタイプを示すアイコンを表示します。表示速度を速くするために、あるいは**タイプ表示**オプションを使用することにして、このオプションをオフにすることもできます。
- **テーブル＆フィールド並び替え**:テーブルおよびフィールドをそれぞれ対応するリストの中で強制的にアルファベット順に表示します。
- **整数を16進数で表示**:通常、数字は10進法で表示されます。このオプションを使用すると、数字が16進法表記で表示されます。注:数値を16進法で入力するには、0x(ゼロの後にx)とタイプし、その後に16進数を続けます。
- **動作状況モニタリングを有効にする**: 動作のモニタリング (アプリケーション内部の詳細チェック) を有効にし、追加されたテーマ (**スケジューラ**、**Web**、**ネットワーク**) に取得した情報を表示します。

次の図はウォッチエリアですべてのオプションが選択された状態を示しています:

メソッド連鎖エリア

1つのメソッドから他のメソッドが呼び出される場合があります。さらにそれらがその他のメソッドを呼び出す場合もあります。このため、デバッグ処理中には、メソッドの連鎖、つまり**呼び出しチェーン**を表示しておくと非常に便利です。メソッド連鎖エリアはデバッグウィンドウの上部右側にあり、この便利な機能を提供しています。
このエリアは、階層リストを使用して表示されます。次の図は、メソッド連鎖エリアの例を示しています:



Call Chain

M_BitTestDemo	
$0	Undefined
DE_LInitialize	
$0	0
$1	->[Customers]
$2	->[Customers]Company
$3	Z
DE_DebugDemo	

- それぞれのメインレベルの項目は、メソッドの名前です。最も上にある項目は、現在トレースしているメソッド、次のメインレベルの項目は呼び出し元のメソッド（現在トレースしているメソッドを呼び出したメソッド）、その次の項目は呼び出し元のメソッドの呼び出し元メソッド、等のように続きます。この例では、メソッドM_BitTestDemoがトレースされています。このメソッドは、メソッドDE_LInitializeによって呼び出され、これはDE_DebugDemoによって呼び出されています。

- メソッド連鎖エリアにあるメソッドの名前をダブルクリックすると、呼び出し元のメソッドのソースコードがソースコードエリアに表示されます。（実行ポイントは移動しません。）
  このようにすると、呼び出し元のメソッドが呼び出されたメソッドへの呼び出しをどのように実行したか、すばやく確認することができます。このようにして、呼び出し連鎖のあらゆる段階を検証することができます。

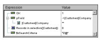
- メソッド名の隣にあるノードアイコンをクリックすると、メソッドのパラメータ（$1,$2...）およびオプションの関数結果（$0）のリストが拡張、または縮小されます。値はエリアの右側に表示されます。右側の矢印にある値をクリックすると、パラメータや関数の結果の値を変更することができます。この図では、以下の通りです:
  1. M_BitTestDemoメソッドは、パラメータを受け取っていません。
  2. M_BitTestDemoメソッドの$0は現在未定義です。これは、メソッドが$0に値を割り当てていないためです（メソッドがこの割り当てをまだ実行していないか、メソッドがサブルーチンであり、関数ではないことが原因です）。
  3. DE_LInitializeメソッドは、DE_DebugDemoメソッドから3つのパラメータを受け取ります。$1は[Customers]テーブルへのポインタ、$2は[Customers]Companyフィールドへのポインタ、$3は値が"Z"の英数字パラメータです。
- メソッドのパラメータリストを展開すれば、パラメータや関数の結果をカスタムウォッチエリアにドラッグ＆ドロップすることができます。

カスタムウォッチエリア

メソッド連鎖エリアのすぐ下にあるのは、**カスタムウォッチエリア**です。このエリアは、式を評価するために使用します。フィールド、変数、ポインタ、演算、組み込み関数、カスタム定義関数等、値を戻すものなら何でも、どのようなタイプの式でも評価できます。

テキスト形式で表示できる式であれば、どのような式でも評価することができます。ピクチャやBLOBのフィールドおよび変数は表示できません。しかし、デバッガは、配列やポインタを階層リストを使用して表示することができます。BLOBの内容を表示するには、BLOB to textのようなBLOBコマンドを使用してください。

次の例では、2つの変数、1つのフィールドポインタ変数、4D関数の結果、演算の項目等が表示されています。



Expression	Value
OK	1
pField	->[Customer]Company
[Customer]Company	""
Records in selection([Customer])	0
$vSearchCriteria	"Z@"

新しい式の挿入

次のようにカスタムウォッチエリアに式を追加して、評価することができます:

- ウォッチエリアからオブジェクトまたは式をドラッグ＆ドロップします。
- メソッド連鎖エリアからオブジェクトまたは式をドラッグ＆ドロップします。
- ソースコードエリアで、評価できる式をクリックします。

空の式を作成するには、カスタムウォッチエリアの任意の空いているスペースをダブルクリックします。すると、新しい式が追加された後、編集モードになり、編集することができます。結果を返す形式の4Dフォーミュラ（変数、フィールド、関数、演算式など）を入力できます。

フォーミュラを入力した後、**enter**または**return**キーを押して（またはエリアの任意の位置をクリックして）、式を評価します。

式を変更するには、その式をクリックして選択し、再びクリックすると（またはテンキー上のenterを押す）編集モードになります。

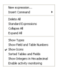
式が必要でなくなった場合には、その式をクリックして選択し、**Backspace**または**delete**キーを押します。

警告: 式を変更した時、後に続くメソッドの実行に影響することに注意してください。特にシステム変数（例えばOK変数）の変更には注意してください。

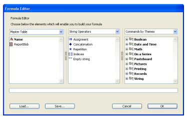
カスタムウォッチエリアのコンテキストメニュー

式を入力し、編集する場合には、カスタム・ウォッチエリアのコンテキストメニューを使用すると、4Dのフォーミュラエディタにアクセスできて便利です。実際には、このコンテキストメニューには、他にもオプションがあります。

このメニューを表示するには、マウスの右ボタンを使用して、**カスタムウォッチエリア**の任意の場所をクリックする。


New expression...
Insert Command
Delete All
Standard Expressions
Collapse All
Expand All
Show Types
Show Field and Table Numbers
✓ Show Icons
Sorted Tables and Fields
Show Integers in Hexadecimal
Enable activity monitoring

- **新規式**：新しい式を入力し、（以下の図のような）4Dのフォーミュラエディタを表示して、新しい式を編集できるようにします。


Formula Editor
Formula Editor
Choose below the elements which will enable you to build your formula
Master Table
String Operators
Commands by Themes
Name
ReportBlob
Assignment
Concatenation
Repetition
Indexes
Empty string
Boolean
Date and Time
Math
On a Series
Pasteboard
Pictures
Printing
Records
String
Load...
Save...
Cancel
OK

フォーミュラエディタに関する詳細は、4D Design Referenceを参照してください。

- **コマンド挿入**：この階層メニュー項目は、フォーミュラエディタを使用せずにコマンドを新しい式として挿入するためのショートカットです。
- **すべて削除**：現在あるすべての式を削除します。
- **標準の式**: 式エリアのオブジェクトリストをコピーします。
- **すべて縮める／すべて拡げる**：階層型リストを使用して評価が実行されたすべての式（ポインタ、配列等）を縮小、または拡大します。
- **タイプ表示**：（適切な場合に）各オブジェクトのオブジェクトタイプを表示します。
- **フィールド＆テーブル番号表示**：フィールドおよびテーブルのそれぞれのフィールド番号、テーブル番号を表示します。TableやFieldを使用してテーブル番号やフィールド番号、ポインタを用いた作業を行っている場合、このオプションは非常に便利です。
- **アイコン表示**：各オブジェクトのオブジェクトタイプを示すアイコンを表示します。表示速度を速くするために、このオプションをオフにすることもできます。その代わりに**タイプ表示**オプションを使用することも出来ます。
- **テーブル＆フィールド並べ替え**：テーブルおよびフィールドをそれぞれ対応するリストの中で強制的にアルファベット順に表示します。
- **整数を16進数で表示**：通常、数字は10進法で表示されます。このオプションを使用すると、数字が16進法表記で表示されます。
  注：数値を16進法で入力するには、0x（ゼロの後にx）とタイプし、その後に16進数を続けます。
- **動作状況モニタリングを有効にする**: モニタした動作状況情報を表示します (ウォッチエリアを参照)。

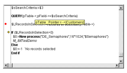
ソースコードエリア

ソースコードエリアには、トレースされているメソッドのソースコードが表示されます。

メソッドが長すぎてテキストエリアに収まらない場合には、スクロールするとメソッドの他の部分も表示できます。

評価できる式（フィールド、変数、ポインター、配列等）にマウスポインタを移動すると、ツールチップが表示され、オブジェクトや式の現在の値とその宣言型が示されます。

次の図はソースコードエリアを示しています:

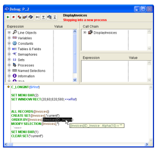


```
$sSearchCriteria:=$3

QUERY(pTable->;pField->=$sSearchCriteria)

$LRecordsInSe[illegible]ble->)

If ($LRecordsInSelection>0)
  $0:=New process("DE_Semaphores";16*1024;"$Semaphores")
  M_BitTestDemo
Else
  $0:=-1  `No records selected
End if
```

pTable : Pointer = ->[Customers]

ツールチップが表示されているのは、マウスポインターが変数pTableの上に置かれているためで、その表示によると、この変数は[Customers]テーブルへのポインターです。

ソースコードエリア内でテキストの部分を選択できます。この場合、選択されたテキストの上にカーソルを移動すると、選択されたオブジェクトの値をTipsとして表示します:



4D Debug: P_2
DisplayInvoices
Stepping into a new process
Expression
Value
Line Objects
Variables
Constants
Tables & Fields
Semaphores
Sets
Processes
Named Selections
Information
Web
Call Chain
DisplayInvoices
Expression
Value

```
C_LONGINT($Wref)

SET MENU BAR(2)
SET WINDOW RECT(20;60;820;580;<>wRef)


ALL RECORDS([Invoices])
CREATE SET([Invoices];"current")
ORDER BY([Invoices];[Invoices]ID_Invoice)
MODIFY SELECTION([Invoices];*)
 `<==>
SET MENU BAR(1)
CLEAR SET("current")
```

[Invoices]ID_Invoice : Alpha(10) = ""

変数名またはフィールドをクリックすると自動的に選択されます。

Tip: **ソースコードエリア**で選択した式（評価できる）をカスタムウォッチエリアにコピーすることが出来ます。次の方法のいずれかでコピーすることが出来ます:

- 単純にドラッグ＆ドロップを行なう（選択したテキストを評価エリア内へドラッグ＆ドロップする）
- Windowsでは**Ctrl+D**、Macintoshでは**command+D**のキーのコンビネーション。

プログラムカウンタ

ソースコードエリアの左マージンにある黄色の矢印（上図を参照してください）は、実行される次の行を表しています。この矢印は、**プログラムカウンター**と呼ばれます。プログラムカウンターは、常に実行寸前の行を表示します。

デバッグのために、メソッド連鎖のトップにあるメソッド（現在実行されているメソッド）のプログラムカウンターの位置を**変更**できます。そのためには、黄色の矢印をクリックして目的の行まで上下にドラッグします。

警告: この機能は、十分注意して使用してください。

プログラムカウンターを順方向に移動しても、スキップした行をデバッガーがすばやく実行されるわけではありません。同様に、プログラムカウンターを逆方向に移動しても、既に実行された行の結果をデバッガーが逆方向に実行されるわけではありません。

プログラムカウンターを移動するということは、デバッガーにその位置からのトレースや実行を追跡するように指示するというだけに過ぎません。すべての現在の設定内容、フィールド、変数等には、プログラムカウンターの移動による影響はありません。

プログラムカウンターを移動する例は、次の通りです。例えば、次のようなコードをデバッグすると仮定します:

```
 ` ...
If(This condition)
 ` DO SOMETHING
Else
 ` DO SOMETHING ELSE
End if
 ` ...
```

プログラムカウンターは、行**If (This condition)**に設定されています。1ステップ先に進むと、プログラムカウンターが行**DO SOMETHING ELSE**に移動していることがわかります。別の分岐にある行を実行しようとしていたため、これは思いがけないことです。この場合には、**This condition**という式が次のステップに影響を与えるような演算を実行していなければ、プログラムカウンタを行**DO SOMETHING**に戻します。これで、コードの当初目的としていた部分を続けてトレースすることができます。

デバッガーへのブレークポイント設定

デバッグの過程において、コードのトレースを一部スキップすることが必要な場合があります。デバッガでは、**特定の位置まで**コードを実行する方法がいくつか提供されます:

- ステップごとの処理中に、**ステップ（呼び出しメソッドもトレース）**ボタンではなく**ステップ（同一メソッドのみ）**ボタンをクリックすることができます。プログラムカウンタ行で呼び出されているサブルーチンや関数の実行を避けたい場合には、この方法が役立ちます。
- 間違ってサブルーチンの処理を始めてしまった場合には、**ステップアウト（呼び出し元へ戻る）**ボタンをクリックすると、そのサブルーチンを実行した後直接、呼び出し元のメソッドに戻ることができます。
- TRACEコマンド呼び出しをある位置で指定した場合には、**トレースなし**ボタンをクリックすると、そのTRACEコマンド呼び出しまでの実行を再開することができます。

次に、プログラムカウンタを行**ALL RECORDS([This Table])**に設定し、次のようなコードを実行していると想定してみます:

```
 ` ...
ALL RECORDS([ThisTable])
$vrResult:=0
For($vlRecord;1;Records in selection([ThisTable]))
   $vrResult:=This Function([ThisTable])
   NEXT RECORD([ThisTable])
End for
If($vrResult>=$vrLimitValue)
 ` ...
```

ここでの目的は、Forループが終了した後で$vrResultの値を評価することです。コードがこの位置まで実行されるにはかなりの処理時間がかかるため、現在の実行をアボートしたくない場合には、TRACEコマンド呼び出しを行**If($vrResult..**の前に挿入するようにメソッドを編集する必要があります。

ループのステップ処理を実行することも1つの方法ですが、[ThisTable]テーブルに何百件ものレコードが入っている場合には、この処理にかなりの時間を費やすことになります。このような状況では、デバッガの**ブレークポイント**を使用できます。ブレークポイントは、ソースコードエリアの左マージンをクリックすると挿入できます。

例題:
次の例では、行**If($vrResult...**のレベルでソースコードエリアの左マージンをクリックします:

このようにすると、その行にブレークポイントが挿入されます。ブレークポイントは赤色の点で表されます。次に、**トレースなし**ボタンをクリックします。

このようにすると、ブレークポイントで示された行まで、通常の実行が再開されます。ブレークポイントで示された行は実行されずに、トレースモードへ 戻ります。この例では、ループ全体は連続して正常に実行されています。そのため、ブレークポイントに到達した時には、マウスボタンを$vrResultの上に移動して、その値をループの終了地点で評価する必要があります。

プログラムカウンタより下方（後に実行される）の行にブレークポイントを設定し、**トレースなし**ボタンをクリックすると、ブレークポイントまでのメソッドをスキップすることができます。

Note: メソッドエディタで直接ブレークポイントを設定することが出来ます。詳しくはブレークポイントを参照してください。

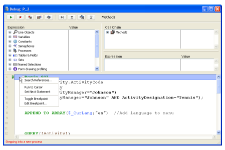
ブレークポイントを追加すると、それはメソッドに保存されます。データベースを再起動してもそこにあります。

永続的ブレークポイントを取り除く方法は、次の2通りです:

- 永続的ブレークポイントを使用した後、赤色の点をクリックすると、ブレークポイントは消えます。
- 永続的ブレークポイントをまだ使用する場合には、これを残しておきたい場合もあります。永続的ブレークポイントを編集すると、一時的に使用不可能にすることができます。編集方法については、ブレークポイントで説明しています。

ソースコードエリアのコンテキストメニュー

ソースコードエリア のコンテキストメニューを使用して、トレースモードでメソッドを実行する際に便利な機能にアクセスできます:



Debug: P_2
Method2
Expression
Value
Line Objects
Variables
Constants
Semaphores
Processes
Tables & Fields
Sets
Named Selections
Form drawing profiling
Call Chain
Method2
Expression
Value
Search References...
Run to Cursor
Set Next Statement
Toggle Breakpoint
Edit Breakpoint...
Stepping into a new process

```
Begin SQL
ity.ActivityCode
y
ityManager="Johnson")
yManager="Johnson" AND ActivityDesignation="Tennis");


APPEND TO ARRAY($_CurLang;"en")  //Add language to menu



QUERY([Activity])
```

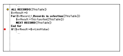
- **定義に移動**: 選択されたオブジェクトの定義に移動する。このコマンドは以下のオブジェクトに使用できます:
  - *プロジェクトメソッド*: 新しいメソッドエディターウィンドウにメソッドの内容を表示
  - *フィールド*: ストラクチャーウィンドウのインスペクターにフィールドプロパティを表示
  - *テーブル*: ストラクチャーウィンドウのインスペクターにテーブルプロパティを表示
  - *フォーム*: フォームエディターにフォームを表示
  - *変数* (ローカル, プロセス, インタープロセス, $n引数): カレントメソッド内で宣言行を表示する、または変数が宣言されたコンパイラーメソッドを表示する
- **参照を検索** (メソッドエディターでも利用可能): メソッド現在の要素が参照されているすべてのデータベースオブジェクト (メソッドとフォーム) を検索。現在の要素とは選択されているものまたはカーソルが存在するものをいいます。これにはフィールド、変数、コマンド、文字列等が含まれます。検索結果は検索結果ウィンドウに表示されます。
- **ここまで実行**: プログラムカウンター (黄色の矢印) とメソッド中で選択された行までの間のコードを実行する。
- **次の行までジャンプ**: 現在の行および途中の行を実行せずに、プログラムカウンターを選択された行まで移動する。選択された行は、ユーザーが実行ボタンのいずれかをクリックした際に実行されます。
- **ブレークポイント切り替え** (メソッドエディターでも利用可能): 選択された行のブレークポイントの有無を切り替えます。これによりメソッドエディターのブレークポイントの有無も切り替わります。
- **ブレークポイント編集** (メソッドエディターでも利用可能): ブレークポイントプロパティダイアログボックスを表示します。ここで行われた変更はメソッドエディターにも反映されます。

ブレークポイント

ソースコードエリアで説明しているように、ブレークポイントは、ブレークしたいコード行と同じレベルで、ソースコードエリアの左マージンかメソッドエディターウィンドウをクリックして設定します。

注: デバッガーのソースコードエリアおよびメソッドエディター内で直接ブレークポイントの挿入、変更、削除をすることができるので、ブレークポイントについてメソッドエディターとデバッガーは、ランタイムエクスプローラーと同様、お互いに影響し合っています。

次の図では、ブレークポイントは**If($vrResult>=$vrLimitValue)**に設定されています:



```
   ` ...
ALL RECORDS([ThisTable])
$vrResult:=0
For($vrRecord;1;Records in selection([ThisTable]))
    $vrResult:=This Function([ThisTable])
    NEXT RECORD([ThisTable])
End for
If ($vrResult>=$vrLimitValue)
   ` ...
```

赤色の点を再びクリックすると、ブレークポイントは削除されます。

ブレークポイント編集

ソースコードエリアのコンテキストメニュー中の**ブレークポイント編集**またはまたはソースコードエリアウィンドウの左マージンでAlt+クリック(Windows)またはoption+クリック(Macintosh)を実行すると、**ブレークポイントプロパティ**ウィンドウにアクセスできます。

- 既存のブレークポイントをクリックすると、そのブレークポイントについてのウィンドウが表示されます。
- ブレークポイントが設定されていない行をクリックすると、デバッガはブレークポイントを作成し、新しく作成されたブレークポイントに関するウィンドウを表示します。

ブレークポイントプロパティウィンドウは、次の図の通りです:

プロパティは、次の通りです:

場所: メソッドの名前とブレークポイントが設定されている行番号を示します。この情報を変更することはできません。

以下に当てはまる場合にブレークする: TrueまたはFalseを返す4Dフォーミュラを入力することによって、条件付きブレークポイントを作成することができます。
例えば、Records in selection([aTable])=0の条件を満たす場合に限った行でブレークさせる場合には、このフォーミュラを入力すると、デバッガがこのブレークポイントを検出した際に、テーブル[aTable]のレコードが選択されていない場合に限ってブレークが発生します。フォーミュラの構文が不確かな場合には、**構文検査**ボタンをクリックします。

ブレーク前のスキップ回数:ループ構造(While、Repeat、For)内、またはループから呼び出されているサブルーチンや関数内のコード行にブレークポイントを設定することができます。例えば、現在調査している問題は、少なくともループを200回繰り返すまでは発生しないことがわかっているものとします。このような場合には200と入力すると、ブレークポイントは201回目の繰り返しからアクティブになります。

ブレークポイント無効:ブレークポイントが現在は必要でないものの、後で必要になるかもしれない場合には、ブレークポイントを編集して一時的に使用不可能にしておくことができます。無効にされたブレークポイントは、デバッグウィンドウのソースコードエリアおよびメソッドエディタ内、ランタイムエクスプローラにおいて、点(・)ではなくダッシュ記号(-)で表示されます。

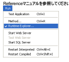
デバッグやメソッドエディタウィンドウ内でブレークポイントを作成、または編集することができます。さらにランタイムエクスプローラのブレークページを使用して、既存のブレークポイントを編集することもできます。詳細についてはブレークリストを参照してください。

ブレークリスト

ブレークリストはデバッガーウィンドウ又はメソッドエディターで作成したブレークポイントを管理することが出来るランタイムエクスプローラのページです。

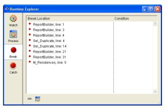
ブレークリストのページを開くには:

1. **実行**メニューから**ランタイムエクスプローラー**を選択する。
ランタイムエクスプローラーは、常に前面に表示されるフローティングパレットに表示させることができます。そうするには、**実行**メニューから**ランタイムエクスプローラー**を選択する際に、**Shift** キーを押してください。結果ランタイムエクスプローラーはすべてのモードで利用可能になります。詳細はDesign Referenceマニュアルを参照してください。


Run
Test Application Ctrl+I
Method... Ctrl+R
✓ Runtime Explorer...
Start Web Server
Test Web Server
Start SQL Server
Restart Interpreted Ctrl+Alt+I
Restart Compiled Ctrl+Shift+I

ランタイムエクスプローラーウィンドウが表示されます。

2. **ブレーク**ボタンをクリックして、ブレークリストを表示させます:


Runtime Explorer
Watch
Process
Break
Catch

Break Location	Condition
ReportBuilder, line: 1	
ReportBuilder, line: 3	
ReportBuilder, line: 4	
Sel_Duplicate, line: 4	
Sel_Duplicate, line: 14	
ReportBuilder, line: 21	
ReportBuilder, line: 31	
M_Residences, line: 5	

ブレークリストは2つの欄から構成されています:

- 左の欄には、ブレークポイントの有効／無効状況と、メソッド名とブレークポイントが設定されている行番号(デバッガウィンドウ又はメソッドエディターを使用)が表示されます。
- 右の欄には、ブレークポイントに関連する条件(ある場合)が表示されます。

このウィンドウを使用して、以下のことが可能です:

- ブレークポイントの条件を設定する。
- それぞれブレークポイントを有効、無効又は削除する。
- ブレークポイントをダブルクリックして定義されたメソッドが表示しているメソッドエディターウィンドウを開く。

しかしウィンドウから新しいブレークポイントを追加することはできません。ブレークポイントはデバッガウィンドウかメソッドエディタからのみ設定できます。

ブレークポイントの条件の設定

ブレークポイントの条件を設定するには、以下のように行います:

1. 右欄をクリックする。

2. ブール値を返す4Dフォーミュラ(式、またはコマンドコールやプロジェクトメソッド)を入力する。

Note: 条件を削除するにはフォーミュラを削除します。

ブレークポイントの有効/無効

ブレークポイントを有効または無効に設定するには:

1. ブレークポイントリストをクリックしてまたは (選択中のブレークが編集モードでなければ) 矢印キーを使用して選択します。

2. コンテキストメニューから**有効**/**無効**を選択する。

ショートカット: ブレークポイントリストの左側の赤い点の上をクリックします。クリック毎に有効/無効が切り替わります。無効になると、赤い点がダッシュ(-)記号になります。

ブレークポイントを削除する

ブレークポイントを削除するには:

1. 項目をクリック、または (現在選択しているコマンド名が編集モードになっていない場合) 矢印キーを使用してリストを選択する。
2. **Delete**または**Backspace**キーを押す、**またはリスト下の削除**ボタンをクリックする。

注: ブレークポイントをすべて削除するには、**すべて削除**ボタン (リスト下部の2番目のボタン) をクリックするか、またはコンテキストメニューから**すべて削除**を選択します。

コマンドのキャッチ

キャッチコマンドリストは、4Dのコマンド呼び出しを捕捉し、デバッガウィンドウを表示するよう指示することが出来るランタイムエクスプローラのページです。

キャッチコマンドは、実行中の全てのプロセスで直ちに効果を発揮し、以降の指定したコマンド呼び出しが行われるとトレース（デバッガウィンドウを表示）に入ります。
ブレークポイントと異なり、特定のメソッドに限らず、全ての全てのプロセス、メソッドが対象となります。

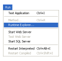
キャッチコマンド（コマンド捕捉）は、任意の場所にブレークポイントを設定することなく、大きな範囲でトレースを行える便利な方法です。例えば、いくつかのプロセスを実行した後に、削除すべきでないレコードが削除されてしまった場合には、DELETE RECORDやDELETE SELECTIONといったコマンドの処理をキャッチ（捕捉）することにより、調査の範囲を狭めることができます。調査対象のコマンドが呼び出されるたびに、デバッガが起動されるので、問題のレコードが削除されてよいかどうかを調べ、コードの誤った箇所を突き止めることができます。

少し経験を積めば、ブレークポイントとコマンドの中断とを組み合わせて使用できるようになります。

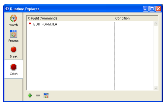
キャッチコマンドページを開くには:

1. **実行** メニューから**ランタイムエクスプローラ**を選択する。

   ランタイムエクスプローラは、フローティングパレットとして表示することができます。この場合、フローティングパレットにすると、常に前面に表示されます。これを行うには**Shift**キーを押しながら、**実行** メニューから**ランタイムエクスプローラ**を選択します。詳細はDesign Referenceマニュアルを参照してください。

   
   Run
   Test Application Ctrl+I
   Method... Ctrl+R
   Runtime Explorer...
   Start Web Server
   Test Web Server
   Start SQL Server
   Restart Interpreted Ctrl+Alt+I
   Restart Compiled Ctrl+Shift+I

   ランタイムエクスプローラウィンドウが表示されます。
2. キャッチボタンをクリックすると、コマンドキャッチリストが表示されます:

   
   Runtime Explorer
   Watch
   Process
   Break
   Catch
   Caught Commands
   Condition

   EDIT FORMULA

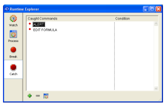
   このページは実行中にトレースに入るコマンドをリスト表示します。2つの欄から構成されています。

- 左の欄には、キャッチするコマンドの有効/無効状況と、コマンド名が表示されます。
- 右の欄には、コマンドに関連する条件（ある場合）が表示されます。

キャッチするコマンドを新しく追加する

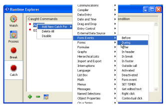
新しくコマンドを追加するには:

1. リストの下部にある追加ボタン（＋）をクリックする。
   または
   キャッチコマンドリスト内でダブルクリックする
   いずれの場合も、ALERTコマンドをデフォルトとして新しいエントリが追加されます。
   エントリが編集モードになります。

   
   Runtime Explorer
   Watch
   Process
   Break
   Catch
   Caught Commands
   Condition

   ALERT
   EDIT FORMULA

2. キャッチしたいコマンド名を入力する
3. **Enter**または**Return**を押して、選択を有効にする
   または
   右マウスボタンをクリックして、コンテキストメニューを表示する。
4. **新規キャッチ追加**を選択し、コマンドテーマ、そしてコマンドを選択します。選択したコマンドがエントリとして追加されます。

   
   Runtime Explorer
   Watch
   Process
   Break
   Catch
   Caught Commands
   ondition
   Add New Catch For
   Delete All
   Disable
   Communications
   Compiler
   Data Entry
   Date and Time
   Drag and Drop
   Entry Control
   External Data Source
   Form Events
   Forms
   Formulas
   Graphs
   Hierarchical Lists
   Import and Export
   Interruptions
   Language
   List Box
   Math
   Menus
   Messages
   Named Selections
   Object Properties
   On a Series
   Before
   During
   After
   In header
   In break
   In footer
   Outside call
   Activated
   Deactivated
   Form event
   SET TIMER
   Get edited text
   Right click
   Contextual click

キャッチするコマンド名を編集する

キャッチコマンドの名前を編集するには:

1. 項目をクリック、または矢印キーを使用してリストを移動する（現在選択しているコマンド名が編集モードになっていない場合）。
2. 編集モードと選択モードとを切り替えるには、**Enter**キーまたは**Return**キーを押す。
3. コマンド名を入力または修正する。
4. 変更を有効にするには、Enterキーまたは Returnキーを押す。入力した名前が既存の4Dコマンドではない場合、項目は修正前の値に設定されます。新しく追加した場合はALERTコマンドに戻ります。

キャッチするコマンドを無効/有効にする

キャッチコマンドを無効、あるいは有効にするには:

1. 項目をクリック、または矢印キーを使用してリストを移動する（現在選択しているコマンド名が編集モードになっていない場合）。
2. 項目が編集モードである場合、enterキーまたはreturnキーを押して選択モードに切り替える。
3. コンテキストメニューから**有効/無効**を選択する。

ショートカット: リストの各項目は、点（丸）をクリックして無効、あるいは有効にすることができます。無効になった場合には、点がダッシュ(-)に変わります。

キャッチするコマンドを削除する

キャッチコマンドを削除するには:

1. 項目をクリック、または矢印キーを使用してリストを移動する（現在選択しているコマンド名が編集モードになっていない場合）。
2. 項目が編集モードである場合、enterキーまたはreturnキーを押して選択モードに切り替える。
3. **Delete**キーを押す、またはリスト下部の削除ボタン (—) を押す。

Note: キャッチコマンドをすべて削除するには、**すべて削除**ボタン (リスト下部3番目のボタン) をクリックするか、またはコンテキストメニューから**すべて削除**を選択します。

キャッチするコマンドに条件を設定する

キャッチコマンドに条件を設定するには:

1. エントリの右の欄をクリックする。
2. ブール値を返す4Dフォーミュラ（式、またはコマンド呼び出しやプロジェクトメソッド）を入力する。

Note: 条件を削除するにはフォーミュラを削除します。

Tips

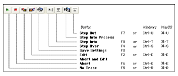
- キャッチコマンドに条件を追加すると、処理速度が低下します。これは、例外条件になるたびに条件が評価されるためです。一方で、条件を追加することにより、デバッグ作業そのものは速く片付きます。これは、条件を満たさない場合は自動的にスキップされるためです。
- キャッチコマンドを無効にすると、削除するのと同じ効果があります。実行中、デバッガはこの項目に対して時間をほとんど割きません。項目を無効にする利点は、再度その項目が必要となった場合に再作成しなくてもいいという点です。

デバッガのショートカット

この節ではデバッガウィンドウで提供されているすべてのショートカットをリストしています。

実行制御ツールバー

次の図ではデバッガウィンドウの左上隅にある9個のボタンのショートカットを示しています:


Button
Windows
MacOS
Step Out　F7　or　Ctrl-U　⌘-U
Step Into Process
Step Into　F8　or　Ctrl-T　⌘-T
Step Over　F4　or　Ctrl-S　⌘-S
Save Settings　F3
Edit　F2　or　Ctrl-E　⌘-E
Abort and Edit
Abort　F6　or　Ctrl-K　⌘-K
No Trace　F5　or　Ctrl-R　⌘-R

Shift＋F5、またはShiftを押しながら**トレースなし**ボタンをクリックすると、実行を再開始します。さらに、現在のプロセスの次のTRACEコマンド呼び出しをすべて無視します。

ウォッチエリア

- ウォッチエリアでマウスの右ボタンをクリック（Windows）またはControl+クリック（Macintosh）すると、ウォッチのコンテキストニューがプルダウンされます。
- ウォッチエリア内の項目をダブルクリックすると、その項目がカスタムウォッチエリアにコピーされます。

メソッド連鎖エリア

- メソッド連鎖エリアでメソッドの名前をダブルクリックすると、ソースコードエリアにそのメソッドのソースが呼び出し行とともに表示されます。

カスタムウォッチエリア

- カスタムウォッチエリアでマウスの右ボタンをクリック（Windows）するか、またはControl+クリック（Macintosh）を実行すると、カスタムウォッチのコンテキストメニューがプルダウンされます。
- カスタムウォッチエリア内でダブルクリックすると、新しい行が作成されます。

ソースコードエリア

- 左マージンをクリックすると、ブレークポイントが設定される（永続的ブレークポイントの場合）、またはブレークポイントが削除されます。
- Alt+Shift+クリック（Windows）またはOption+Shift+クリック（Macintosh）により、一時的ブレークポイントが設定されます。
- Alt+クリック（Windows）またはOption+クリック（Macintosh）により、**ブレーク編集**ウィンドウが表示されます。
- 式やオブジェクトを選択し、カスタムウォッチエリアにドラッグ&ドロップすると、コピーされます。
- 選択したテキスト上でCtrl (Windows) または Command (Mac OS) キーを押しながらクリックすると、カスタムウォッチエリアにテキストがコピーされます。

すべてのエリア

- Ctl+*(Windows)またはコマンド＋*(Mac OS)を押すと、ウォッチエリアが更新されます。
- どのエリアでも項目が選択されていない場合に**Enter**キーを押すと、1行ずつ進みます。
- 項目の値が選択されている場合には、矢印キーを使用してリスト内を移動します。
- 項目が編集されている場合には、矢印キーを使用するとカーソルが移動します。Ctrl+A/X/C/V（Windows）またはCommand+A/X/C/V（Macintosh）を編集メニューのすべてを選択/切り取り/コピー/貼り付けメニューへのショートカットとして使用できます。

4D環境

- Application file
- Application type
- Application version
- BUILD APPLICATION
- CHECK MEMORY LEAKS
- Compact data file
- COMPONENT LIST
- CREATE DATA FILE
- Data file
- FLUSH BUFFERS
- Get 4D folder
- Get database localization
- Get database parameter
- Get last update log path New 14.0
- Get memory leaks
- GET MEMORY STATISTICS Updated 14.0
- GET SERIAL INFORMATION
- Get table fragmentation
- Is compiled mode
- Is data file locked
- NOTIFY RESOURCES FOLDER MODIFICATION
- OPEN ADMINISTRATION WINDOW
- OPEN DATA FILE
- OPEN SECURITY CENTER
- OPEN SETTINGS WINDOW
- PLUGIN LIST
- QUIT 4D
- RESTART 4D New 14.0
- SET DATABASE LOCALIZATION
- SET DATABASE PARAMETER Updated 14.0
- SET UPDATE FOLDER New 14.0
- Structure file
- VERIFY CURRENT DATA FILE
- VERIFY DATA FILE
- Version type Updated 14.0
- ADD DATA SEGMENT
- DATA SEGMENT LIST

Application file

Application file -> 戻り値

引数	型		説明
戻り値	文字	↩	4D実行形式のファイルまたはアプリケーションのパス名

説明

Application fileコマンドは、現在使用している4D実行形式のファイルまたはアプリケーションのパス名を返します。

Windows
例えば、ボリュームE上の\PROGRAMS\4Dに配置された4Dを使用している場合、この関数は、E:\PROGRAMS\4D\4D.EXEを返します。

Macintosh
例えば、Macintosh HDディスク上のProgramsフォルダの中にある4Dを使用している場合、この関数は、Macintosh HD:Programs:4D.appを返します。

例題

Windows上で、4Dの起動時に、DLLライブラリが4D実行形式のファイルと同じ階層に配置されているかどうかをチェックする必要があるとします。On Startupデータベースメソッドに次のコードを記述します:

```
If(On Windows & (Application type#4D Server))
   If(Test path name(Long name to path name(Application file)+"XRAYCAPT.DLL")#Is a document)
  ` XRAYCAPT.DLLがない旨の警告を表示する
  ` つまりX-ray capture 機能は使用できません
   End if
End if
```

注: プロジェクトメソッドOn WindowsとLong name to path nameはシステムドキュメントで説明しています。

Application type

Application type -> 戻り値

引数	型	説明
戻り値	倍長整数	➡ アプリケーションタイプを示す数値

説明

Application type コマンドは、現在実行している4D環境のアプリケーションタイプを示す数値を返します。4Dは、以下のようにあらかじめ定義された定数を持っています:

定数	型	値
4D Local Mode	倍長整数	0
4D Remote Mode	倍長整数	4
4D Server	倍長整数	5
4D Volume Desktop	倍長整数	1

例題

On Server Startupデータベースメソッド以外のいずれかの箇所で、4D Serverを実行中かチェックする必要がある場合、以下のように記述できます:

```
If(Application type=4D Server)
  ` 適切な処理を行う
End if
```

Application version

Application version {(buildNum {; *})} -> 戻り値

引数	型		説明
buildNum	倍長整数	←	ビルド番号
*	演算子	→	指定した場合、ロングバージョン番号 指定しなかった場合、ショートバージョン番号
戻り値	文字	↩	バージョン番号のエンコードされた文字列

説明

Application version コマンドは、現在使用している4D環境のバージョン番号を表すエンコードされた文字列を返します。

引数オプション*を指定しない場合、以下のようにフォーマットされた4文字の文字列を返します。

文字位置	説明
1-2	バージョン番号
3	アップデート番号
4	リビジョン番号

例：文字列"0600"は、"version 6.0.0."を表します。

引数オプション*を指定した場合、以下のようにフォーマットされた8文字の文字列を返します。

文字位置	説明
1	"F" は最終バージョン "B" はベータバージョン それ以外の文字は、4Dの内部バージョンを意味する
2-3-4	4Dの内部コンパイル番号
5-6	バージョン番号
7	アップデート番号
8	リビジョン番号

例：文字列"B0120602"は、"version 6.0.2.Beta 12"を表します。

Application version コマンドはオプションのbuildNum 引数に追加の情報、お使いの4Dのビルド番号を返すことができます。これは内部的なコンパイル番号で、4Dの技術チームにお使いの4Dの環境について伝える際に使用できます。

注: 4D Volume Licenceがマージされたアプリケーションにおいては 返される "build" 番号に意味はありません。この状況ではバージョン情報は開発者により管理されます。

例題 1

次の例は、4D環境のバージョン番号を表示します。

```
$vs4Dversion:=Application version
ALERT("使用しているバージョンは "+String(Num(Substring($vs4Dversion;1;2)))+"."+
$vs4Dversion[[3]]+"."+$vs4Dversion[[4]])
```

例題 2

以下の例は、最終版の4Dを使用しているかどうかを検査します。

```
If(Substring(Application version(*);1;1)#"F")
   ALERT("製品版の4Dおよびデータベースを使用しているかを確認してください。")
   QUIT 4D
End if
```

BUILD APPLICATION

BUILD APPLICATION {(projectName)}

引数	型		説明
projectName	文字	→	使用するプロジェクトのフルアクセスパス

説明

BUILD APPLICATION コマンドはアプリケーションの生成処理を開始します。処理時にはカレントのアプリケーションプロジェクトの設定、またはprojectName 引数で渡したアプリケーションプロジェクト設定が使用されます。

アプリケーションプロジェクトはXMLファイルで、アプリケーションを生成するために使用されるすべてのパラメターが含まれています。ほとんどのパラメターはアプリケーションビルドダイアログボックスで確認できます。詳細は4D Design Referenceマニュアルの を参照してください。
デフォルトで、4Dはデータベースごとに"buildapp.prj"という名前のアプリケーションプロジェクトを作成し、データベースのPreferencesフォルダー内、BuildAppサブフォルダーに配置します。

データベースがコンパイルされていないか、コンパイルされたコードが古い場合、コマンドはまずコンパイル処理を起動します。この場合、エラーが発生しない限り、コンパイラウィンドウは表示されません。進捗バーのみが表示されます。
この進捗バーは、MESSAGES OFF コマンドを使用することによって非表示にすることができます。

オプションのprojectName 引数を渡さない場合、コマンドは標準のファイルを開くダイアログボックスを表示し、プロジェクトファイルの選択を要求します。ダイアログボックスが受け入れられると、システム変数Documentに開かれたプロジェクトファイルのフルパスが格納されます。
有 効なアプリケーションプロジェクトとして、XMLファイル (UTF-8エンコーディングおよび拡張子".xml") のアクセスパスと名前を渡すと、コマンドはファイルで定義されたパラメターを使用します。アプリケーションプロジェクトのXMLファイルで利用可能な構造 とキーに関する詳細はアプリケーションビルド設定ファイルを参照してください。

例題

この例題では、1つのメソッドで2つのアプリケーションを作成します:

```
BUILD APPLICATION("c:\\folder\\projects\\myproject1.xml")
If(OK=1)
   BUILD APPLICATION("c:\\folder\\projects\\myproject2.xml")
End if
```

システム変数またはセット

コマンドが正しく実行されると、システム変数OKに1が、そうでなければ0が設定されます。システム変数Documentには開かれたプロジェクトファイルのフルパス名が格納されます。

エラー処理

コマンドの実行に失敗すると、ON ERR CALL コマンドで割り込み可能なエラーが生成されます。

CHECK MEMORY LEAKS

CHECK MEMORY LEAKS
このコマンドは引数を必要としません

説明

このコマンドは内部使用のために予約されており、使用されるべきではありません。

Compact data file

Compact data file (structurePath ; dataPath {; archiveFolder {; option {; method}}}) -> 戻り値

引数	型	説明
structurePath	テキスト	→ ストラクチャファイルのパス名
dataPath	テキスト	→ 圧縮するデータファイルのパス名
archiveFolder	テキスト	→ オリジナルのデータファイルを置く、フォルダのパス名
option	倍長整数	→ 圧縮オプション
method	テキスト	→ 4Dコールバックメソッド名
戻り値	テキスト	↩ 元のデータファイルが置かれたフォルダの完全パス名

説明

Compact data fileコマンドは、ストラクチャーstructurePathに関連付けられている、dataPath 引数で指定されたデータファイルを圧縮します。圧縮に関する詳細は4D Design Referenceマニュアルを参照してください。

データベースの操作の継続を確かなものにするため、圧縮された新しいデータファイルが自動で元のファイルと置き換えられます。安全のため、元のファイルは変更されず、"Replaced files (compacting) YYYY-MM-DD HH-MM-SS"という特別なフォルダに移動されます。YYYY-MM-DD HH-MM-SSはバックアップが行われた日付と時刻を表します。例えば"Replaced files (compacting) 2007-09-27 15-20-35"のようになります。

コマンドは、元のデータファイルを格納するために作成されたフォルダの、実際の完全なパス名を返します。このコマンドはローカルモードの4D、または4D Serverのストアドプロシージャでのみ実行できます。圧縮するデータファイルは、structurePathで指定するストラクチャファイルに対応するものでなければなりません。さらにコマンド実行時にデータファイルが開かれていてはなりません。そうでなければエラーが生成されます。
圧縮処理中にエラーが発生した場合、元のファイルが最初の場所に保持されます。インデックスファイル (.4DIndxファイル) がデータファイルに関連付けられていれば、それも圧縮されます。データファイルと同様元のファイルは保存され、新しく圧縮されたファイルと置き換えられます。

- structurePath 引数には、圧縮するデータファイルに関連付けられたストラクチャファイルの完全パス名を渡します。この情報は圧縮プロシージャのために必要です。パス名はOSのシンタックスで表現されなければなりません。なお空の文字列を渡すと、標準のファイルを開くダイアログボックスが表示され、使用するストラクチャファイルを選択させることができます。
- dataPath 引数には、空の文字列、ファイル名、またはOSのシンタックスで表現された完全パス名を渡すことができます。空の文字列を渡すと、標準のファイルを開くダイアログボックスが表示され、圧縮するデータファイルを選択させることができます。このファイルはstructurePath 引数で指定されたストラクチャファイルに対応するものでなければなりません。データファイル名のみを渡すと、4Dはストラクチャファイルと同階層でデータファイルを探します。
- オプションのarchiveFolder 引数を使用して、元のデータファイルとインデックスファイルを保存する"Replaced files (compacting) DateTime"フォルダの場所を指定できます。
  このコマンドは、実際に作成されたこのフォルダのパス名を返します。
  - この引数を省略すると、元のファイルは自動で、ストラクチャファイルと同階層に作成される"Replaced files (compacting) DateTime"フォルダに置かれます。
  - 空の文字列を渡すと、標準のフォルダを開くダイアログが表示され、ユーザは作成するフォルダの場所を選択できます。
  - OSのシンタックスを使用してパス名を指定すると、コマンドは指定された場所に"Replaced files (compacting) DateTime"フォルダを作成します。
- オプションのoptions 引数を使用して、さまざまな圧縮オプションを指定できます。これを行うにはData File Maintenanceテーマの以下の定数を使用してください。加算することで複数のオプションを指定できます:

定数	型	値	コメント
Compact address table	倍長整数	131072	強制的にレコードのアドレステーブルを更新します (圧縮時間は長くなります)。このオプションのみを指定すると、4Dは自動でレコードの再保存オプションを有効にします。この場合、レコード番号が更新される点に留意してください。
Create process	倍長整数	32768	このオプションが渡されると圧縮は非同期で行われ、コールバックメソッドを使用して結果を管理しなければなりません。4Dは進捗状況を表示しません (コールバックメソッドを使用して表示させることができます)。プロセスが正しく起動されるとOKシステム変数が1に設定され、他の場合は0に設定されます。このオプションが渡されない時、圧縮が行われればOK変数に1が設定され、そうでなければ0が設定されます。
Do not create log file	倍長整数	16384	通常このコマンドはXMLフォーマットのログファイルを作成します。このオプションを使用すればログファイルは作成されません。
Update records	倍長整数	65536	現在のストラクチャー定義に基づき、すべてのレコードを強制的に再保存します。

- method 引数は、Create processオプションを渡したときに、圧縮中定期的に呼び出されるコールバックメソッドを指定するために使用します。このオプションが指定されていなければ、コールバックメソッドが呼び出されることはありません。このメソッドに関する詳細は、VERIFY DATA FILEコマンドの説明を参照してください。
  コールバックメソッドがデータベースに存在しない場合、エラーが生成され、OKシステム変数に0が設定されます。

デフォルトで、Compact data fileコマンドはXMLフォーマットのログファイルを作成します (Do not create log fileオプションを指定しない場合。options引数の説明を参照)。名称はデータファイルに基づき、元のデータファイルと同じ階層に作成されます。例えば、データファイルの名称が"data.4dd,"であるとき、ログファイルの名称は"data_compact_log.xml"となります。

例題

以下の例題 (Windows) は、データファイルの圧縮を実行します:

```
$structFile:=Structure file
$dataFile:="C:\Databases\Invoices\January\Invoices.4dd"
$origFile:="C:\Databases\Invoices\Archives\January\"
$archFolder:=Compact data file($structFile;$dataFile;$origFile)
```

システム変数およびセット

圧縮処理が正しく終了したら、OKシステム変数に1が設定されます。そうでなければ0が設定されます。

COMPONENT LIST

COMPONENT LIST (componentsArray)

引数	型	説明
componentsArray	文字配列	← コンポーネント名

説明

COMPONENT LISTコマンドは、4Dがカレントのホストデータベースにロードしたコンポーネントの名前を、componentsArray 配列に返します。

データベースが開かれると、4DはComponentsフォルダ内の有効なコンポーネントをロードします。

- 4Dアプリケーション実行ファイルと同階層にあるComponentsフォルダー
- ストラクチャーファイルと同階層にあるComponentsフォルダー

注：両方に同じコンポーネントがインストールされている場合、4DはストラクチャーCと同階層のCOmponentsフォルダーにあるコンポーネントをロードします。

このコマンドは、ホストデータベースまたはコンポーネントから呼び出すことができます。データベースがコンポーネントを使用しない場合、componentsArray 配列は空となります。

コンポーネントの名前は、マトリクスデータベース (.4db, .4dc または .4dbase) のストラクチャー名です。このコマンドを使用して、コンポーネントがインストールされているかいないかにより追加の機能を提供する、アーキテクチャやモジュールインタフェースを設定できます。

4Dコンポーネントに関する詳細は、Design Referenceマニュアルを参照してください。

CREATE DATA FILE

CREATE DATA FILE (accessPath)

引数	型		説明
accessPath	文字	→	作成するデータファイルの名前または完全パス名

説明

CREATE DATA FILEコマンドは、オンザフライで新しいデータファイルをディスク上に作成し、4Dアプリケーションで開かれているデータファイルと置き換えます。

このコマンドの動作は、OPEN DATA FILEコマンドと同じです。唯一の相違点は、ストラクチャファイルを再オープンした後に、accessPath引数で指定された新しいデータファイルを作成することです。

処理を開始する前に、コマンドは指定されたアクセスパスが既存のファイルに該当していないかどうかを調べます。

4D Server: 4D v13より、このコマンドを4D Serverで実行することができるようになりました。この場合コマンドは、指定されたファイルを作成する前に、まずサーバー上でQUIT 4Dを呼び出します (結果各リモートマシン上にサーバーが終了する旨通知するダイアログボックスが表示されます)。

Data file

Data file {(segment)} -> 戻り値

引数	型		説明
segment	倍長整数	→	廃止、使用されません
戻り値	文字	↩	データベースのデータファイルのロング名

説明

Data file コマンドは、現在使用しているデータベースのデータファイルのロング名を返します。

4Dバージョン11より、セグメントはサポートされなくなりました。segment 引数は無視され、使用されません。

Windows上
例えば、ボリュームG上の\DOCS\MyCDsに配置されたデータベースMyCDsを使って作業している場合、この関数は
G:\DOCS\MyCDs\MyCDs.4DDを返します（データベース作成時にデフォルトの場所と名前を使用した場合）。

Macintosh上
例えば、ハードディスクMacintosh HD上のDocuments:MyCDsフォルダに配置されたデータベースを使って作業している場合、この関数は
Macintosh HD:Documents:MyCDs:MyCDs.4DDを返します（データベース作成時にデフォルトの場所と名前を使用した場合）。

警告：リモートモードの4Dからこのコマンドを呼び出した場合、ロング名ではなくデータファイル名のみが返されます。

FLUSH BUFFERS

FLUSH BUFFERS
このコマンドは引数を必要としません

説明

FLUSH BUFFERSコマンドを実行すると、即座にデータバッファの内容をディスクに保存します。データベースへのすべての変更をディスクに保存します。

通常、4Dがデータの変更内容を定期的に保存するため、このコマンドを呼び出す必要はありません。環境設定のデータ管理ページにある、**データをディスクに保存 : X分毎** オプションを使用すると、バッファのフラッシュ間隔を指定できます。

Note: 4Dは、I/O操作を高速化するために内蔵のデータキャッシュシステムを使用します。データ更新はいくらかの時間データキャッシュに置かれ、ディスクには保存されません。しかしこのことは開発者のコードに対し透過的です。例えばQUERYを呼び出した場合、4Dのデータベースエンジンは、データキャッシュを検索して結果を出します。

Get 4D folder

Get 4D folder ({folder}{;}{*}) -> 戻り値

引数	型		説明
folder	倍長整数	→	フォルダタイプ (省略時 = Active 4D folder)
*	演算子	→	ホストデータベースのフォルダを返す
戻り値	文字	←	指定したフォルダのパス名

説明

Get 4D folderコマンドは、カレントアプリケーションのアクティブな4Dフォルダー、またはfolder引数で指定された4D環境フォルダーへのパス名を返します。このコマンドを使用して、4Dアプリケーションで使用されるフォルダーの実際のパス名を取得できます。このコマンドを使用すれば、記述したコードがローカライズされたどのシステムのプラットフォームでも動作することが保障されます。

folderには4D Environmentテーマの次のいずれかの定数を渡します:

定数	型	値
4D Client Database Folder	倍長整数	3
Active 4D Folder	倍長整数	0
Current Resources Folder	倍長整数	6
Database Folder	倍長整数	4
Database Folder Unix Syntax	倍長整数	5
Extras Folder	倍長整数	2
HTML Root Folder	倍長整数	8
Licenses Folder	倍長整数	1
Logs Folder	倍長整数	7

各フォルダーについて以下で説明します:

フォルダー名に関する予備的な説明:

- {Disk} はシステムがインストールされたディスクを示します。
- Userという単語は、セッションを開いたユーザーの名前を示します。

Active 4D Folder

4D環境は以下の情報を保存するために**4D**フォルダーを使用します:

- 4D環境アプリケーション、ツール、およびユティリティプログラムが使用する環境設定ファイル
- Shortcuts.xml ファイル (カスタマイズされたキーボードショートカット)
- Macros v2 フォルダー (メソッドエディターで使用するマクロコマンド)
- Favorites v11、Favorites v12およびFavorites v13フォルダー (開いたローカルおよびリモートデータベースのパス名)

デフォルトで4Dフォルダーは以下の場所に作成されます:

- Windows Vista/Windows 7: {Disk}:\Users\Current user\AppData\Roaming\4D
- Windows XP: {Disk}:\Documents and Settings\Current user\Application Data\4D
- Mac OS: {Disk}:Users:Current user:Library:Application Support:4D

Licenses Folder

マシンのライセンスファイルを含むフォルダーです。

Licensesフォルダーは以下の場所に作成されます:

- Windows Vista/Windows 7: {Disk}:\ProgramData\4D\Licenses
- Windows XP: {Disk}:\Documents and Settings\All Users\Application Data\4D\Licenses
- Mac OS: {Disk}:Library:Application Support:4D:Licenses

注:

- 4D Volume Desktopとマージされたアプリケーションの場合、Licensesフォルダーはアプリケーションのパッケージに含まれます。
- 権限が足りないためLicensesフォルダーがシステムに作成できない場合、フォルダーは以下の場所に作成されます:
  - Windows Vista/Windows 7: {Disk}:\Users\Current user\AppData\Roaming\4D\Licenses
  - Windows XP : {Disk}:\Documents and Settings\Current user\Application Data\4D\Licenses
  - Mac OS : {Disk}:Users:Current user:Library:Application Support:4D:Licenses

Extras Folder (廃止)

クライアントマシン毎にダウンロードされる、カスタマイズされたコンテンツを格納するフォルダー。

互換性に関する注意: 4D v11 SQLのバージョン11.2より、サーバとリモートマシン間でカスタマイズされたファイルの交換を行うためのExtrasフォルダーの利用は推奨されなくなりました。この目的ではResourcesフォルダーの利用を推奨します (以下の Current Resources Folderの説明参照)。
Extras フォルダーは既存のアプリケーションの互換性を保つために、4D Serverによりサポートされます。

注: データベースにExtrasフォルダーが存在しない場合、**Get 4D folder**コマンドにExtras Folder定数を渡して実行すると、フォルダーが作成されます。

4D Client Database Folder

各4Dクライアントマシンに作成された4Dデータベースフォルダ。リーソス、プラグイン、Resourcesフォルダー等データベースに関連したファイルやフォルダーを格納するためのフォルダです。

4D Client Database Folderはそれぞれのクライアントマシン上の以下の場所に置かれます:

- Windows Vista/Windows 7: *{Disk}:\Users\Current user\AppData\Local\4D\DatabaseName_Address*
- Windows XP: *{Disk}:\Documents and Settings\Current user\Local Settings\Application Data\4D\DatabaseName_Address*
- Mac OS: *{Disk}:Users:User:Library:Caches:4D:DatabaseName_Address:*

Database Folder

データベースストラクチャファイルを含むフォルダーです。パス名は、現在のプラットホームの標準のシンタックスを使用して表現されます。
4Dクライアントアプリケーションの場合、この定数は前述の4D Client Database Folder定数とまったく同じです。コマンドはローカルに作成されたフォルダーのパス名を返します。

Database Folder Unix Syntax

データベースストラクチャファイルを含むフォルダーです。この定数は前述のフォルダーと同じものですが、パス名は/Users/...のようなUNIXシンタックス(POSIX) で表現されます。このシンタックスは主に、Mac OSでLAUNCH EXTERNAL PROCESSコマンドを使用する、または_o_SET CGI EXECUTABLEコマンドを使用する場合に、使用されます。

Current Resources folder

データベースのリソースフォルダー。このフォルダーにはデータベースのインタフェースで使用される、ピクチャーやテキストなどの追加の項目が置かれます。コンポーネントはそれぞれ独自のResourcesフォルダーを持ちます。Resourcesフォルダーはデータベースストラクチャーと同階層に置かれます。
クライアント/サーバーモードでは、サーバーマシンとクライアントマシン間でピクチャーやファイル、サブフォルダーなどのカスタムデータを交換するために使用できます。このフォルダーのコンテンツはクライアントマシンが接続するたびに自動で更新されます。Resourcesフォルダーに関連付けられたすべての参照メカニズムは、クライアント/サーバーモードでもサポートされます (.lproj フォルダー、XLIFF、ピクチャーなど)。さらに4DではリソースエクスプローラーなどさまざまなツールTを通じ、このフォルダの管理と更新を動的に行えるようになっています。

注: データベースにResourcesフォルダーが存在しない場合、**Get 4D folder**コマンドにCurrent Resources folder定数を渡して実行すると、フォルダーが作成されます。

Logs folder

データベースのLogsフォルダー。このフォルダーにはカレントデータベースのログが格納されます。フォルダーはストラクチャファイルと同階層に作成され、以下のログが格納されます:

- データベース変換
- Webサーバリクエスト
- データ検証と修復
- ストラクチャ検証と修復
- バックアップ/復元処理のジャーナル
- コマンドデバッグ
- 4D Serverリクエスト (クライアントマシンとサーバー上で生成)

注: データベースにLogsフォルダーが存在しない場合、**Get 4D folder**コマンドにLogs Folder定数を渡して実行すると、フォルダーが作成されます。

HTML Root Folder

データベースのカレントHTMLルートフォルダー。返されるパス名は、プラットフォームの標準シンタックスで表現されます。HTMLルートフォルダーは、リクエストされたページやファイルを4D Webサーバが探すフォルダーです。デフォルトで、このフォルダーの名前は**WebFolder**で、ストラクチャーファイルと同階層 (またはリモートモードの4Dの場合、そのローカルコピー) に置かれます。この場所はデータベース設定のWeb/設定ページ、またはWEB SET ROOT FOLDERコマンドで動的に変更できます。
Get 4D folderコマンドがリモートの4Dが呼ばれた場合、返されるパスはリモートマシンのものです。4D Serverのものではありません。

オプションの * 引数は、コンポーネントを使用するアーキテクチャにおいて有用です。ホストデータベースとコンポーネント、どちらのフォルダーのパス名を取得するか指定するために使用します。この引数はDatabase Folder、Database Folder、Unix Syntax、Current Resources folderのみで使用できます。他の場合、この引数は無視されます。
このコマンドがコンポーネントから呼び出されると:

- * 引数が渡されていれば、コマンドはホストデータベースのフォルダーパス名を返します。
- * 引数が渡されていなければ、コマンドはコンポーネントのフォルダーパス名を返します。

返されるデータベースフォルダー (Database FolderとDatabase Folder Unix Syntax) はコンポーネントのアーキテクチャにより異なります:
- .4dbase フォルダー/パッケージの場合、コマンドは.4dbase フォルダー/パッケージのパス名を返します。
- .4dbまたは.4dcファイルの場合、コマンドは"Components"フォルダーのパス名を返します。
- エイリアスやショートカットの場合、コマンドはオリジナルのマトリクスデータベースが格納されているフォルダーのパス名を返します。結果は、先に説明したとおり、このデータベースのフォーマット (.4dbase フォルダー/パッケージ、または.4db/.4dcファイル) により異なります。
コマンドがホストデータベースから呼ばれた場合、* 引数が渡されているかどうかにかかわらず、コマンドは常にホストデータベースのフォルダーのパス名を返します。

例題 1

シングルユーザの4Dで、起動時に、4Dフォルダにある設定ファイルを読み込み (または作成) したいとします。これを行うために、に以下のように記述できます:

```
MAP FILE TYPES("PREF";"PRF";"Preferences file")
  ` PREF Mac OSファイルタイプと.PRF Windowsファイル拡張子をマップ
$vsPrefDocName:=Get 4D folder+"MyPrefs" ` 環境設定ファイルへのパス名
  ` ファイルが存在するかチェック
If(Test path name($vsPrefDocName+(".PRF"*Num(On Windows)))#Is a document)
   $vtPrefDocRef:=Create document($vsPrefDocName;"PREF") ` なければ作成
Else
   $vtPrefDocRef:=Open document($vsPrefDocName;"PREF") ` あれば開く
End if
If(OK=1)
  ` ドキュメントの中身を処理
   CLOSE DOCUMENT($vtPrefDocRef)
Else
  ` エラー処理
End if
```

例題 2

以下の例は、Mac OS上でDatabase Folder Unix Syntax定数を使用し、データベースフォルダの内容を取り出します:

```
$posixpath:="\""+Get 4D folder(Database Folder Unix Syntax)+"\""
$myfolder:="ls -l "+$posixpath
$in:=""
$out:=""
$err:=""
LAUNCH EXTERNAL PROCESS($myfolder;$in;$out;$err)
```

Note: Mac OSでは、スペースがファイルやフォルダ名に含まれる場合、パス名をクォートで括らなければなりません。文字列にクォートを挿入するために、エスケープ文字"\"を使用できます。あるいはChar(Double quote)を使用することもできます。

システム変数およびセット

folder 引数が無効または返されたパス名が空の場合、OKシステム変数は0に設定されます。

Get database localization

Get database localization {(languageType)} -> 戻り値

引数	型		説明
languageType	倍長整数	→	Type of language
戻り値	文字	↩	データベースのカレントランゲージ

説明

Get database localizationコマンドは、デフォルトランゲージまたはlanguageTypeで指定されたデータベースの言語を、RFC 3066で定義された標準で返します。例えばコマンドは英語の場合"en"を、日本語の場合"ja"を返します。この標準およびコマンドから返される値の情報については、*Design Reference*マニュアルの付録 C: XLIFFアーキテクチャーを参照してください。

アプリケーション内では同時に複数の異なる言語設定を使用できます。取得する設定を指定するには、languageTypeに4D Environmentテーマの以下の定数を渡します。

定数	型	値	コメント
Current localization	倍長整数	1	アプリケーションのカレント言語: デフォルト言語またはSET DATABASE LOCALIZATIONコマンドで設定された言語。
Default localization	倍長整数	0	Resourcesフォルダとシステム環境に基づき、4Dが起動時に自動で設定する言語 (変更不可)。
Internal 4D localization	倍長整数	3	並び替えやテキスト比較で4Dが使用する言語 (アプリケーションの環境設定で設定)。
User system localization	倍長整数	2	システムのカレントユーザーが設定した言語

languageTypeを省略するとデフォルトでコマンドはデフォルトランゲージ (0) を返します。

データベースのカレントランゲージは、ローカライズされたアイテムをプログラムが検索する.projフォルダを指定するために使用されます。4Dは自動で、データベースの開始時に、システム環境と**Resources**フォルダの内容で、カレントのランゲージを決定します。4Dは以下のような優先順位で、参照ランゲージに対応する.lprojフォルダを読み込みます:

1. システムランゲージ (Mac OSでは、環境設定で複数のランゲージの順番を設定できます。4Dはこの設定を使用します)。
2. 4Dアプリケーションのランゲージ
3. English
4. **Resources**フォルダで最初に見つかったフォルダ

Note: データベースに.lprojフォルダがない場合、4Dは以下の優先順位を適用します。1. システムランゲージ、2. English (システムランゲージを決定できなかった場合)

Get database parameter

Get database parameter ({aTable ;} selector {; stringValue}) -> 戻り値

引数	型	説明
aTable	テーブル	→ パラメタを取得するテーブル、または 引数が省略された場合デフォルトテーブル
selector	倍長整数	→ データベースパラメタコード
stringValue	文字	← パラメタの文字列値
戻り値	実数	↩ パラメタの現在の値

説明

Get database parameterコマンドを使用して、現在の4Dデータベースパラメーターの値を知ることができます。パラメーター値が文字列の場合、それはstringValue 引数に返されます。

selector 引数には、知りたいパラメーターを指定します。4DはDatabase Parametersテーマで、定義済み定数を提供しています:

定数	型	値	コメント
Direct2D Disabled	倍長整数	0	セレクター69 (Direct2D Status) 参照。
Direct2D Hardware	倍長整数	1	セレクター69 (Direct2D Status) 参照。
Seq Order Ratio	倍長整数	1	*** このセレクターは廃止されました ***
Direct2D Hardware SVG and Editors	倍長整数	2	セレクター69 (Direct2D Status) 参照。
Seq Access Optimization	倍長整数	2	*** このセレクターは廃止されました ***
Direct2D Software	倍長整数	3	セレクター69 (Direct2D Status) 参照。
Seq Distinct Values Ratio	倍長整数	3	*** このセレクターは廃止されました ***
Direct2D Software SVG and Editors	倍長整数	4	セレクター69 (Direct2D Status) 参照。
Index Compacting	倍長整数	4	*** このセレクターは廃止されました ***
Direct2D Mixed	倍長整数	5	セレクター69 (Direct2D Status) 参照。
Seq Query Select Ratio	倍長整数	5	*** このセレクターは廃止されました ***
Minimum Web Process	倍長整数	6	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **とりうる値**: 0 -> 32 767 **説明**: ローカルモードの4Dならびに4D Serverを使用した場合に、非コンテキストモードで保持するWebプロセスの最小数。デフォルト値は0（下記参照）。
Maximum Web Process	倍長整数	7	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **とりうる値**: 0 -> 32 767 **説明**: ローカルモードの4Dならびに4D Serverを使用した場合に、非コンテキストモードで保持するWebプロセスの最大数。デフォルト値は10。 非コンテキストモードでのWebサーバの反応を良くするため、4Dは5秒間Webプロセスを遅延させ、将来やってくるかもしれないHTTPクエリを処理する際、それを再利用します。パフォーマンスの観点では、クエリごとに新しいプロセスを作成するよりも、再利用したほうが実際に有利です。Webプロセスが再利用されると、さらに5秒間遅延されます。Webプロセスが最大数に達するとプロセスがアボートされます。5秒の遅延以内にWebプロセスがクエリを受け取らない場合、Webプロセスの最小数を下回らなければ、プロセスはアボートされます。最小数未満になる場合、プロセスは再度遅延されます。 これらの引数は、リクエスト数や利用可能なメモリ、その他のパラメタに応じて、Webサーバの動作を調整できるようにするものです。
Web Conversion Mode	倍長整数	8	**スコープ**: カレントプロセス **2セッション間で設定を保持**: No **とりうる値**: 0, 1, 2 または 3 • (デフォルトモード) ブラウザが対応する場合、HTML 4.0フォーマットに変換。そうでなければHTML 3.2フォーマットと配列を使用。 • 1 = 6.0.x変換モード • 2 = 6.5変換モード • 3 = HTML 4.0フォーマット + CSS-P (バージョン6.5.2より) に変換。 **説明**: ローカルモードの4Dならびに4D Serverで使用するWeb用の4Dフォーム変換モード。デフォルトで、4D WebサーバはCSS1を使用し、4Dで表示される4Dフォームと同様のHTMLページを生成します。この機能では、バージョン6.7より前の4Dで作成されたデータベースに関しては、フォームが正しく変換されない可能性があります。このため、フォーム変換モードを設定する必要があるかもしれません。 このモードは、SET DATABASE PARAMETERが呼び出されたプロセス（Webコンテキスト）に対してのみ設定されます。このコマンドは、メソッド内で呼び出してデータベースのすべてのフォームを確実に統一したり、あるいは特定のフォームを表示する前にのみ呼び出すこともできます。このコマンドは、コンテストモード、またはWebプロセス以外の場所から呼び出すと、何も行いません。
Database Cache Size	倍長整数	9	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: - **説明**: カレントのデータベースメモリキャッシュサイズを取得するために使用します。戻り値はバイト単位です。 最大キャッシュサイズは環境設定の"データベース/データ管理"ページで設定できます。しかしデータベースキャッシュに実際に割り当てられる値は設定とシステムの現在のリソースに基づきます。このセレクタを使用すれば4Dによりデータベースに割り当てられた実際のメモリサイズを取得できます。 **警告**: ランゲージを使用してデータベースキャッシュメモリサイズを設定することはできません。言い換えればDatabase Cache SizeをSET DATABASE PARAMETERコマンドで使用することはできません。
4D Local Mode Scheduler	倍長整数	10	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: Yes **説明**: セレクタ12参照
4D Server Scheduler	倍長整数	11	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: Yes **説明**: セレクタ12参照
4D Remote Mode Scheduler	倍長整数	12	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ10, 11 および 12に対し、value引数は16進数、0x00aabbccの形式で表わされます。詳細は次の通りです: *aa* = システムへのコール毎の最小tick数（0～100） *bb* = システムへのコール毎の最大tick数（0～100） *cc* = システムへのコール間のtick数（0～20） これらの値が範囲外のとき、4Dはその値を最大数に設定します。value引数には、次の定義済標準値のうちいずれかを渡すことができます: • value = -1: 4Dに最高優先度を割り当てる • value = -2: 4Dに平均的な優先度を割り当てる • value = -3: 4Dに最低優先度を割り当てる **説明**: この引数を使用して、4Dシステム内部コールをダイナミックに設定することができます。selectorに応じて、スケジューラの値は次のアプリケーションのために設定されます。 • シングルユーザの4Dから呼び出された場合、ローカルモードの4D (selector=10). • 4D Serverから呼び出された場合、4D Server (selector=11). • 4DServerに接続した4Dから呼び出された場合、リモートモードの4D (selector=12). **Note:** セレクタ=12(4D Remote Mode Scheduler)は、SET DATABASE PARAMETERコマンドがサーバーマシン上で実行されるか、クライアントマシン上で実行されるかにより異なります: • コマンドがサーバーマシン上で実行される場合、新しい値はコマンド実行後にサーバに接続する全てのクライアントマシンに適用されます。 • コマンドがクライアントマシン上で実行される場合、新しい値はそのクライアントマシンで直ちに有効となり、またコマンド実行後にサーバに接続するすべてのクライアントマシンに対し適用されます。 この動作を使用することで、クライアントマシン毎に異なる特性をダイナミックに扱うことが可能です。これはコマンドの実行がまず、そのクライアントマシンの設定に影響し、その後にサーバに接続するすべてのクライアントマシンへの設定値として使用されるという動作に基づきます。 この動作は、4Dの6.8.6, 2003.3と4D2004以降のバージョンで効果があります。 **警告**: これらのセレクタを不適切に使用すると、アプリケーションパフォーマンスが深刻に低下する原因となります。これらの要素に対する完全な知識がある場合のみ変更を行うことをお勧めします。
4D Server Timeout	倍長整数	13	**スコープ**: value が正数なら4Dアプリケーション **2セッション間で設定を保持**: value が正数ならYes **とりうる値**: 0 -> 32 767 **説明**: この引数を使用して、4D Serverのタイムアウトの値を変更することができます。4D Serverのタイムアウトのデフォルト値は、サーバ側の「環境設定」ダイアログボックスの「クライアント-サーバ/接続設定」ページで定義します。 セレクタ4D Server Timeoutにより、対応する引数valueの新しいタイムアウト（分単位で指定）を設定できます。この機能は、クライアント側でCPUを占有する、時間がかかる処理を実行する前に、タイムアウト設定を長くしたい場合は特に便利です。例えば、膨大なページの印刷などは、予期しないタイムアウトになる可能性があります。 2種類のオプションがあります: • value引数に**正数**を渡すと、グローバルかつ永続的なタイムアウトが設定されます。この新しい値はすべてのプロセスに対して適用され、4Dアプリケーションの環境設定に保存されます（環境設定ダイアログボックスで変更した場合と同じ）。 • value引数に**負数**を渡すと、ローカルで一時的なタイムアウトが設定されます。この新しい値は呼び出し元のプロセスに対してのみ適用され（他のプロセスではデフォルトの値を維持）、例えば処理の終了時のように、クライアントが動作していることを示す信号をサーバが受信すると即座に、デフォルト値へリセットされます。このオプションは、4Dプラグインにより開始された時間のかかる処理を管理する際に便利です。 "タイムアウトしない"オプションを設定するには、valueに0 を渡します。
4D Remote Mode Timeout	倍長整数	14	**スコープ**: value が正数の場合4D アプリケーション **2セッション間で設定を保持**: value が正数の場合Yes **説明**: この引数を使用して、4D Serverに接続したリモートの4Dマシンが参照するタイムアウト値を設定できます。リモートモードの4Dが使用するデフォルトのタイムアウト値は、リモートマシン上の環境設定の"クライアント-サーバ/設定"ページで設定されます。 このセレクタに関する詳細は4D Server Timeout (13) の説明を参照してください。4D Remote Mode Timeout セレクタは非常に特殊な状況において使用されます。
Port ID	倍長整数	15	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: No **説明**: この引数を使用して、ローカルモードの4Dおよび4D Serverによる4D Webサーバが使用するTCPポートをオンザフライで変更することができます。デフォルト値は80で、この値は環境設定ダイアログボックスの"Web/設定"ページで設定できます。TCP Port Numbersテーマの定数をvalue引数に使用できます。 Port IDセレクタは、コンパイルして4D Desktopを組み込んだ4D Webサーバで役立ちます（この場合、デザインモードへのアクセス手段がありません）。TCPポートIDに関する詳細はWebサーバー設定を参照してください。
IP Address to listen	倍長整数	16	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Character set	倍長整数	17	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Max Concurrent Web Processes	倍長整数	18	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Client Minimum Web Process	倍長整数	19	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ6参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Maximum Web Process	倍長整数	20	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ7参照 **Description**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Max Web requests size	倍長整数	21	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ27参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Port ID	倍長整数	22	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ15参照 **Description**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client IP Address to listen	倍長整数	23	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ16参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Character set	倍長整数	24	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ17参照 **Description**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Max Concurrent Web Proc	倍長整数	25	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ18参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Cache writing mode	倍長整数	26	*** このセレクターは廃止されました ***
Maximum Web requests size	倍長整数	27	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
4D Server Log Recording	倍長整数	28	**Scope**: 4D Server, 4D リモート **2セッション間で設定を保持**: No **とりうる値**: 0 または 1 から X (0 = 記録しない, 1 から X = ファイル名に付加するシーケンス値)。 **説明**: 4D Serverが受け取る標準的なリクエスト（Webリクエストを除く）の記録を開始または停止します。デフォルト値は0（リクエストを記録しない）です。 サーバマシンが受信した各リクエストをログファイルに記録するよう、4D Serverに指示することができます。このメカニズムが有効になると、データベースストラクチャと同じ階層にログファイルが作成されます。ファイルには"4DRequestsLogX"（Xはログのシーケンシャル番号）の名前が付けられます。ファイルサイズが10MBに達するとそのファイルは閉じられ、インクリメントしたシーケンス番号を使用して新しいファイルが作成されます。もし同じ名前のファイルが存在する場合は置き換えられます。シーケンシャル番号の開始番号を、value引数を使用して指定できます。 このテキストファイルにはそれぞれのリクエストに関する様々な情報（時間、プロセス番号、ユーザ、リクエストサイズ、処理時間など）がシンプルな表形式のフォーマットで保存されます。この情報はアプリケーションの調整を行う場合や統計の目的で利用する場合に特に役立ちます。この情報を、例えばスプレッドシートソフトウェアに読み込んで処理することもできます。 **Note:** マニュアル操作により、ログの記録をOn/Offすることが出来ます。Windowsでは**Ctrl+Alt+L**、MacOSでは**Command+Option+L**のショートカットです。
Web Log Recording	倍長整数	29	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Client Web Log Recording	倍長整数	30	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: 0 = 記録しない (デフォルト), 1 = CLFフォーマットで記録, 2 = DLFフォーマットで記録, 3 = ELFフォーマットで記録, 4 = WLFフォーマットで記録 **説明**: すべてのクライアントマシンのWebサーバが受け取るWebリクエストの記録を開始または停止します。デフォルト値は0（リクエストを記録しない）です。 このセレクタの動作はセレクタ29と同じです。しかし対象はWebサーバとして使用されるすべての4Dクライアントマシンです。この場合、"logweb.txt"ファイルは4Dクライアントデータベースフォルダ (キャッシュフォルダ) のLogsサブフォルダに作成されます。特定のクライアントマシンにのみ設定を行いたい場合、4Dクライアントの環境設定を使用します。
Table Sequence Number	倍長整数	31	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: Yes **とりうる値**: 任意の倍長整数値 **説明**: このセレクタは、引数に渡したテーブルのレコードの、カレントのユニーク番号を取得あるいは設定するために使用します。"カレントの数値"とは"最後に使用された数値"を意味します。SET DATABASE PARAMETERコマンドを使用してこの値を変更すると、渡された値+1の番号を使用して次のレコードが作成されます。この新しい番号は、Sequence numberコマンドによって返される、さらにはストラクチャエディタやSQLで自動インクリメントが設定されたフィールドに返される番号です。 デフォルトで、この固有の番号は4Dが設定し、レコードの作成順に対応します。詳細はSequence numberコマンドのドキュメントを参照してください。
_O_Real display precision	倍長整数	32	*** このセレクターは廃止されました ***
TCP_NODELAY	倍長整数	33	*** このセレクターは廃止されました ***
Debug log recording	倍長整数	34	
Client Server Port ID	倍長整数	35	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0～65535 **説明**: 4D Server が(4D Client に対して）データベースを公開するために使用されるTCPポート番号をプログラムで変更するために使用します。デフォルト値は19813 です。 この値を変更すれば、TCPプロトコルを使用して、複数の4D クライアント/サーバアプリケーションを同じマシンで同時に使用することができます。その場合、それぞれのアプリケーションごとに異なるポート番号を設定します。 公開ポート番号は、ストラクチャファイルに記録されています。ローカルモードの4Dで設定することもできますが、クライアント/サーバ環境でのみ考慮されます。 値を変更した場合、新しい設定が有効になるためには、アプリケーションを再起動する必要があります。
Invert Objects	倍長整数	37	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0, 1 または 2 (0 = モードは無効, 1 = 自動モード, 2 = モードが有効) **説明**: Right-to-left 言語のWindowsでデータベースが表示されるとき、アプリケーションモードでフォームやオブジェクト、メニューバーなどを反転させるために使用される、"オブジェクト反転"モードを設定します。このモードはアプリケーション環境設定のデータベース/インターナショナルページで変更できます。 • 0 に設定した場合、システム設定に関係なく、モードは無効です（環境設定でいいえにするのと同じ）。 • 1 に設定した場合、システム設定に応じ、モードが有効または無効になります（環境設定を自動にするのと同じ）。 • 2 に設定した場合、システム設定に関係なく、モードは有効です（環境設定をはいにするのと同じ）。 詳細は4DのDesign Referenceマニュアルを参照してください。
HTTPS Port ID	倍長整数	39	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Client HTTPS Port ID	倍長整数	40	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: 0 ～ 65535 **説明**: このセレクタは、クライアントマシンのWebサーバがSSLによるセキュアな接続（HTTPSプロトコル）で使用するTCPポート番号を、プログラムで変更するために使用できます。デフォルトの値は443（標準ポート番号）です。 このセレクタの動作はセレクタ39と同じですが、Web サーバとして使用されるすべてのクライアントマシンに適用されます。特定のクライアントマシンの設定だけを変更するのであれば、4Dリモートの環境設定ダイアログ画面を使用してください。
Unicode mode	倍長整数	41	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0 (互換モード) または 1 (Unicodeモード) **説明**: カレントデータベースの文字セットに関連する動作モード。4DはUnicode文字セットをサポートしますが、(Mac ASCII文字セットに基づく)"互換"モードで動作させることができます。デフォルトで、変換されたデータベースは互換モード(0) で、バージョン11以降で作成されたデータベースはUnicodeモードで実行されます。実行モードは環境設定のオプションでコントロールでき、またこのセレクタを使用して読みだしたり、(テスト目的で) 変更したりできます。このオプションを変更した場合、それを有効にするにはデータベースを再起動しなければなりません。コンポーネント内部ではこの値を変更できないことに留意してください。読み出しのみが可能です。
Temporary memory size	倍長整数	42	*** このセレクターは廃止されました ***
SQL Autocommit	倍長整数	43	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0 (無効) または 1 (有効) **説明**: SQLの自動コミットモードを有効または無効にするために使用します。デフォルトは 0 (無効モード) です。 自動コミットモードは、データベースの参照整合性を強化するために使用されます。このモードが有効の時、すべてのSELECT, INSERT, UPDATE そして DELETE (SIUD) クエリは、これらがトランザクション内で実行されていない場合、自動でアドホックなトランザクションに含められます。このモードはデータベースの環境設定でも設定できます。
SQL Engine Case Sensitivity	倍長整数	44	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0 (大文字小文字を考慮しない) または 1 (考慮する) **説明**: SQLエンジンが文字列比較を行う際に、大文字と小文字の違いを考慮させるかどうかを設定します。 デフォルトで値は1 (大文字小文字を考慮する) です。SQLエンジンは文字列比較 (並び替えやクエリ) の際に大文字と小文字を異なる文字として扱います。例えば"ABC"= "ABC"ですが"ABC" # "Abc"です。SQLエンジンと4Dエンジンの動作をそろえたいなど特定の場合では、大文字と小文字を区別しない文字列比較 ("ABC"="Abc") を使用できます。 このオプションはアプリケーション環境設定の SQL/設定ページで設定できます。
Client Log Recording	倍長整数	45	**スコープ**: リモート4Dマシン **2セッション間で設定を保持**: No **とりうる値**: 0 または 1 から X (0 = 記録しない, 1 から X = ファイル名に付加するシーケンス値)。 **説明**: コマンドを実行した4Dクライアントマシンが実行した標準的なリクエスト（Webリクエストを除く）の記録を開始または停止します。デフォルト値は0（リクエストを記録しない）です。 クライアントマシンが実行したリクエストをログファイルに記録するよう、4Dに指示することができます。このメカニズムが有効になると、クライアントマシンのデータベースのローカルフォルダ内、Logsサブフォルダに2つのログファイルが作成されます。ファイルには"4DRequestsLog_X"と"4DRequestsLog_ProcessInfo_X"（Xはログのシーケンシャル番号）の名前が付けられます。4DRequestsLogファイルサイズが10MBに達するとそのファイルは閉じられ、インクリメントしたシーケンス番号を使用して新しいファイルが作成されます。もし同じ名前のファイルが存在する場合は置き換えられます。シーケンシャル番号の開始番号を、value引数を使用して指定できます。 このテキストファイルにはそれぞれのリクエストに関する様々な情報（時間、プロセス番号、ユーザ、リクエストサイズ、処理時間など）がシンプルな表形式のフォーマットで保存されます。この情報はアプリケーション開発フェーズや統計の目的で利用する場合に特に役立ちます。
Query By Formula On Server	倍長整数	46	**スコープ**: カレントテーブルおよびプロセス **2セッション間で設定を保持**: No **とりうる値**: 0 (データベース設定を使用), 1 (クライアント上で実行) または 2 (サーバ上で実行) **説明**: 引数に渡されたtableに対して実行されるQUERY BY FORMULAやQUERY SELECTION BY FORMULAコマンドの実行場所。 クライアント/サーバモードでデータベースを使用するとき、フォーミュラを使用したクエリをサーバ上またはクライアント上で実行させることができます: • 4D v11 SQLで作成されたデータベースでは、これらのコマンドはサーバ上で実行されます。 • 変換されたデータベースでは、これらのコマンドは、以前のバージョンの4Dと同様、クライアントマシン上で実行されます。 • 変換されたデータベースでは、環境設定のアプリケーション/互換性ページで、これらのコマンドの実行場所をグローバルに変更できます。 この実行場所の違いは、アプリケーションのパフォーマンス (通常サーバ上で実行したほうが早い) だけでなく、プログラミングにも影響します。実際フォーミュラの部品の値 (特にメソッドから呼ばれる変数) は、実行コンテキストにより異なります。このセレクタを使用して開発者は、アプリケーションの動作を適応させられます。 value 引数に0を渡すと、フォーミュラを使用するクエリの実行場所は、データベースの設定に基づきます: 4D v11 SQLで作成されたデータベースでは、これらのコマンドはサーバ上で実行されます。変換されたデータベースでは、データベース環境設定に基づき、クライアントマシンまたはサーバマシンで実行されます。valueに1または2を渡すと、これらのコマンドの実行

			場所をクライアントマシンまたはサーバマシンに強制できます。 例題4を参照してください。 **Note:** "SQLタイプ"のJOINを有効にしたい場合 (QUERY BY FORMULA Joins (49) セレクタ参照) は、フォーミュラがレコードにアクセスすることができるよう、常にフォーミュラをサーバ上で実行しなければなりません。このコンテキストでは、フォーミュラはいかなるメソッド呼び出しも含んではいけないことに注意してください。そうでない場合、自動でリモートマシンでの実行に切り替わります。
Order By Formula On Server	倍長整数	47	**スコープ**: カレントテーブルおよびプロセス **2セッション間で設定を保持**: No **とりうる値**: 0 (データベース設定を使用), 1 (クライアント上で実行) または 2 (サーバ上で実行) **説明**: 引数に渡されたtableに対して実行されるORDER BY FORMULAコマンドの実行場所。 クライアント/サーバモードでデータベースを使用するとき、ORDER BY FORMULAコマンドをサーバ上またはクライアント上で実行させることができます。このセレクタを使用して、このコマンドの実行場所 (サーバまたはクライアント) を指定できます。このモードはデータベース環境設定でも設定できます。詳細はセレクタ46、Query By Formula On Serverの説明を参照してください。 **Note:** "SQLタイプ"のJOINを有効にしたい場合 (QUERY BY FORMULA Joins (49) セレクタ参照) は、フォーミュラがレコードにアクセスすることができるよう、常にフォーミュラをサーバ上で実行しなければなりません。このコンテキストでは、フォーミュラはいかなるメソッド呼び出しも含んではいけないことに注意してください。そうでない場合、自動でリモートマシンでの実行に切り替わります。
Auto Synchro Resources Folder	倍長整数	48	**スコープ**: 4D リモートマシン **2セッション間で設定を保持**: No **とりうる値**: 0 (同期しない), 1 (自動同期) または2 (確認する). **説明**: このコマンドを実行する4DクライアントマシンのResourcesフォルダの動的な同期モード。 サーバ上のResourcesフォルダの内容が更新されたり、(リソースエクスプローラや SET DATABASE LOCALIZATION コマンドで) ユーザが同期をリクエストすると、サーバは接続されたユーザに通知を行います。 クライアント側では3つの同期モードを選択できます。Auto Synchro Resources Folderセレクタはカレントセッションでクライアントマシンが使用するモードを指定できます: • 0 (デフォルト値): 動的な同期を行わない (同期リクエストは無視される) • 1: 自動の動的同期 • 2: クライアントマシンにダイアログを表示し、同期の受け入れ回避を確認する アプリケーションの環境設定で、同期モードをグローバルに設定できます。
QUERY BY FORMULA Joins	倍長整数	49	**スコープ**: カレントプロセス **2セッション間で設定を保持**: No **とりうる値**: 0 (データベース設定を使用), 1 (常に自動リレーションを使用) または 2 (可能ならSQL JOINを使用) **説明**: "SQL JOIN"の利用に関連する、QUERY BY FORMULAとQUERY SELECTION BY FORMULAコマンドの動作モード。 4D v11 SQLのバージョン11.2以降で作成されたデータベースでは、これらのコマンドはSQL JOINモデルに基づくJOINを実行します。このメカニズムを使用して、(以前のバージョンでは必要な条件だった) 自動リレーションで接続されていない他のテーブルに対して実行されたクエリに基づき、テーブルのセレクションを変更できます。 QUERY BY FORMULA Joinsセレクタで、カレントプロセスの、フォーミュラによるクエリの動作モードを指定できます: • 0: データベースの現在の設定を使用する (デフォルト値)。4D v11 SQLのバージョン11.2以降で作成されたデータベースでは、"SQL JOIN"はフォーミュラによるクエリに対し常に有効です。変換されたデータベースでは、互換性のためデフォルトでこのメカニズムは有効になっていません。しかし環境設定で実装できます。 • 1: 常に自動リレーションを使用 (= 以前のバージョンの4Dの動作)。このモードでは、他のテーブルに対して行われたクエリに基づきテーブルのセレクションを作成するためには、リレーションが必要です。4Dは"SQL JOIN"を行いません。 • 2: 可能であればSQL JOINを使用 (= 4D v11 SQLのバージョン11.2以降で作成されたデータベースのデフォルト動作)。このモードでは、フォーミュラが適合する限り、4Dはフォーミュラによるクエリに"SQL JOIN"を使用します (2つの例外があります。QUERY BY FORMULAやQUERY SELECTION BY FORMULA コマンドの説明を参照)。 **Note:** 4Dのリモートモードにおいて、"SQL JOIN"はフォーミュラがサーバ上で実行されているときにのみ使用できます (レコードへのアクセスが必要なため)。フォーミュラの実行場所を設定するには、セレクタ46と47を参照してください。
HTTP Compression Level	倍長整数	50	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
HTTP Compression Threshold	倍長整数	51	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Server Base Process Stack Size	倍長整数	53	**スコープ**: 4D Server **2セッション間で設定を保持**: No **とりうる値**:正の倍長整数 **説明**: サーバ上のプリエンプティブシステムプロセス毎に割り当てるスタックのサイズ (バイト単位) です。デフォルトでこの値は1,000,000 (1 MB) です。 プリエンプティブシステムプロセスはメインの4D クライアントプロセスを制御するためにロードされます。デフォルトでそれぞれのプリエンプティブプロセスに割り当てられるサイズはおよその場合最適なサイズですが、何百ものプロセスが作成されるようなケースではこのサイズが適切かどうか検討する必要が出てくるかもしれません。 データベースが実行する処理がそれを許す限り、最適化の目的でこのサイズを大幅に減らすことができます (例えばデータベースで大量のレコードの並び替えなどを行わない場合)。512 や256 KB でさえも設定可能です。 スタックサイズを小さくしすぎることは致命的であり、4D Server の動作に害を及ぼすことになるので注意してください。このパラメタの設定は注意を持って行い、データベースの利用状況 (レコード数や行う処理など) を考慮しながら行わなければなりません。 このパラメタの設定を行うには、On Server Startup データベースメソッドなどを使用してサーバ上でコマンドが実行されなければなりません。
Idle Connections Timeout	倍長整数	54	**スコープ**: 値が負数なら4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**: 持続時間を秒で表す値。値は正数 (新規接続) または負数 (既存の接続)をとることができます。デフォルト値は4D v11 SQLの場合0 (タイムアウトなし) で、4D v12の場合20です。 **説明**: この引数を使用して、4DデータベースエンジンとSQLエンジン両方への動きのない接続の最大時間 (タイムアウト) を設定できます。この設定は、コマンドが実行されたマシンにより開かれたすべての接続に適用されます。 動作していない接続がこの制限時間に達すると、接続は自動でスタンバイ状態に置かれます。つまりクライアント/サーバセッションがフリーズされ、ネットワークソケットが閉じられます。この動作はユーザに対し完全に透過的です。スタンバイ状態の接続でリクエストが開始されると、ソケットが自動で再び開かれ、クライアント/サーバセッションが再び有効になります。 この設定によりサーバのリソースを節約できます。スタンバイ状態の接続はソケットを閉じ、サーバ上のプロセスを解放します。他方これにより、ファイアウォールがアイドルなソケットを閉じてしまうことに伴い接続が失われることを避けることができます。このためには、アイドル接続のタイムアウト値はファイアウォールのタイムアウト値よりも小さくなくてはなりません。 valueに正数を渡すと、設定はすべてのプロセスのすべての新規接続に適用されます。負数を渡すと、設定はカレントプロセスの開かれた接続に適用されます。0を渡すと、アイドル接続のタイムアウトは行われません。 4D v11 SQLの場合、このパラメターはサーバ上でのみ考慮されます。 4D v12の場合、このパラメターはサーバーおよびクライアント両側で設定できます。2つの異なる間隔を設定すると、短いほうが使用されます。通常この値を変更する必要はありません。
PHP Interpreter IP address	倍長整数	55	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: "nnn.nnn.nnn.nnn" (例 "127.0.0.1") のようなフォーマット文字列 **説明**: FastCGI を経由してPHPインタプリタと通信を行うために、4D がローカルで使用するIP アドレス。デフォルトで値は"127.0.0.1" です。このアドレスは4D が配置されているマシンに対応しなければなりません。このパラメタはデータベース設定を使用してすべてのマシン用にグローバルに設定できます PHPインタプリタに関する詳細は*Design Reference*マニュアルを参照してください。
PHP Interpreter port	倍長整数	56	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: 正の倍長整数値。デフォルト値は8002。 **説明**: 4DのPHPインタプリタが使用するTCPポート番号。データベース設定でこの値をすべてのマシン用にグローバルに変更できます。PHPインタープリタに関する詳細は*Design Reference* マニュアルを参照してください。
PHP Number of children	倍長整数	57	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: 正の倍長整数値。デフォルト値は5。 **説明**: 4DのPHPインタプリタがローカルで作成し、管理する子プロセスの数。最適化の目的で、スクリプト実行リクエストを処理するために、PHPインタプリタは" 子プロセス" と呼ばれるシステムプロセスのセット (プール) を作成、使用します。アプリケーションのニーズに基づき、子プロセス数の数を変更できます。データベース設定を使用してこの値をすべてのマシン用にグローバルに設定できます。PHPインタプリタに関する詳細は*Design Reference* マニュアルを参照してください。 **Note:** Mac OS では、すべての子プロセスは同じポートを共有します。Windows では、それぞれの子プロセスが個別のポート番号を使用します。最初の番号はPHP インタプリタ用に設定された番号で、他の子プロセスは最初の番号をインクリメントします。例えばデフォルトポート番号が8002で、5個の子プロセスを起動すると、ポート8002から8006が使用されます。
PHP Max requests	倍長整数	58	**スコープ**: 4D application **2セッション間で設定を保持**: No **値**: 正の倍長整数地。デフォルト値は500。 **説明**: PHP インタプリタが受け入れるリクエストの最大数。この最大値に達すると、インタプリタは"server busy"タイプのエラーを返します。セキュリティおよびパフォーマンスのため、この値を変更できます。データベース設定を使用してすべてのマシン用にグローバルに設定を変更できます。このパラメタに関する詳細はFastCGIPHPのドキュメントを参照してください。 **Note:** 4D側では、これらの引数は動的に適用されます。設定を有効にするために4Dを終了する必要はありません。他方、PHPインタプリタが既に起動されている場合、これらの設定を有効にするためにはインタプリタを再起動しなければなりません。
PHP Use external interpreter	倍長整数	60	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: 0 = 内部インタプリタを使用, 1 = 外部インタプリタを使用 説明: 4D のPHPリクエストを4Dの内部インタプリタに送信するか、外部インタプリタに送信するかを指定する値。デフォルトで値は0 (4Dのインタプリタを使用) です。独自にインストールしたPHPインタプリタを使用したい場合、例えば追加のモジュールを使用したり特定の設定を使用したい場合は、valueに1を渡します。この場合、4DはPHPリクエストの場合でも内部インタプリタを起動しません。 カスタムPHPインタプリタはFastCGIでコンパイルされ、4Dエンジンと同じマシン上に配置されていなければなりません。この場合、開発者がインタプリタを完全に管理しなければならない点に留意してください。4Dはインタプリタを開始したり停止したりしません。 データベース設定を使用してこの値をすべてのマシン用にグローバルに設定できます。
Maximum Temporary Memory Size	倍長整数	61	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**:正の倍長整数値 **説明**: 4D がそれぞれのプロセスに割り当てることのできる一時的なメモリの最大サイズ (MB)。デフォルトで値は 0 (最大サイズの設定なし) です。 4D はインデックスやソート処理のために特別な一時的メモリを使用します。このメモリは大量の処理を行う間、" 標準" キャッシュメモリの保護を意図したものです。これは必要な時にのみ有効になります。デフォルトで、一時的なメモリのサイズは、(システムメモリ設定に基づく) 利用可能なリソースによってのみ制限されます。 このメカニズムはほとんどのアプリケーションで適しています。しかし特定の特別なコンテキスト、特に同時に多数のシーケンシャルソートを行うようなサーバ/ クライアントアプリケーションでは、一時的なメモリのサイズが、システムが不安定になるほどに致命的に増加するかもしれません。このような場合は、一時的メモリの最大サイズを設定することで、アプリケーションが正しく動作するようにできます。その代わり、実行速度に影響が出ます。プロセスに対する最大サイズに達すると、4D はディスクファイルを使用し、そのために処理が遅くなります。 先のようなケースの場合、だいたい50 MB が一般的なサイズとしてよいと思われます。しかし適切な値はアプリケーションの特性、そして実際の環境でのテスト結果に基づき決定されるべきです。
SSL Cipher List	文字列	64	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**: コロンで区切られた一連の文字列 (例 "RC4-MD5:RC4-64-MD5:....") **説明**: 暗号リストはSSLプロトコルのために4Dが使用します。例えばこのセレクターを使用してSSL 3.0暗号化アルゴリズムを実装でき、そしてSSL 2.0による接続を拒否できます。 この設定はアプリケーション全体に適用されます (HTTP サーバーやSQLサーバー、およびSSL プロトコルを使用するすべての4D の機能に関連) が、その効果は一時的であり、セッションをまたいでは保持されません。 暗号リストが変更されると、新しい設定を有効にするために、関連するサーバーを再起動しなければなりません。 暗号リストを (SLIファイルに恒久的に格納された) デフォルト値に再設定するには、value引数に空の文字列 ("") を渡してSET DATABASE PARAMETERコマンドを呼び出します。 デフォルトで、4DはRC4暗号化アルゴリズムを使用します。(より最新の) AESアルゴリズムを使用するには、value引数に以下の文字列を渡します: "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH" **Note**: Get database parameterコマンドで暗号リストはオプションのstringValue引数に返され、戻り値は常に0となります。
Cache unload minimum size	倍長整数	66	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**: 1より大きい正の倍長整数 **説明**: エンジンがオブジェクトをデータベースキャッシュに配置する際に空き空間を作成する必要が出た場合、データベースキャッシュからリリースするメモリの最小サイズ (バイト単位)。 このセレクタの目的はキャッシュからデータをリリースする時間を減らし、よりよりパフォーマンスを得ることにあります。キャッシュのサイズやデータベース中で処理されるデータのブロックサイズに応じてこの値を変更できます。 このセレクタが使用されないとデフォルトで、4Dは空間が必要になった時最低10%のキャッシュをアンロードします。
Direct2D Status	倍長整数	69	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **説明**: WindowsにおけるDirect2D実装のアクティベーションモード **取りうる値**: 以下の定数のいずれか (デフォルトでモード5): Direct2D Disabled (0): Direct2Dモードは無効であり、データベースは過去のモード (GDI/GDIPlus) で動作する。 Direct2D Hardware (1): 4Dアプリケーション全体でDirect2Dグラフィックハードウェアコンテキストを使用する。このコンテキストが利用できない場合、Direct2Dグラフィックソフトウェアコンテキストを使用 (Vistaを除く。VistaではパフォーマンスのためにGDI/GDIPlusモードが使用されます)。 Direct2D Hardware SVG and Editors (2): SVG、コード、およびストラクチャーエディターでDirect2Dグラフィックハードウェアコンテキストを使用する。このコンテキストが利用できない場合、Direct2Dグラフィックソフトウェアコンテキストを使用 (Vistaを除く。VistaではパフォーマンスのためにGDI/GDIPlusモードが使用されます)。 Direct2D Software (3): Windows 7以降、4Dアプリケーション全体でDirect2Dグラフィックソフトウェアコンテキストを利用。VistaではパフォーマンスのためにGDI/GDIPlusモードが使用されます。 Direct2D Software SVG and Editors (4): Windows 7以降、SVG、コード、およびストラクチャーエディターのDirect2Dグラフィックソフトウェアコンテキストを使用する。VistaではパフォーマンスのためにGDI/GDI/GDIPlusモードが使用されます。 Direct2D Mixed (5) (デフォルトモード): Windows 7以降、SVG、コード、およびストラクチャーエディターでDirect2Dグラフィックハードウェアコンテキストを使用する。4Dアプリケーションの他の部分ではDirect2Dグラフィックソフトウェアコンテキストを使用する。VistaではパフォーマンスのためGDI/GDIPlusモードを使用します。

Direct2D Get active status	倍長整数	74	**注:** このセレクターはGet database parameterコマンドでのみ使用することができ、設定することはできません。
			説明: WindowsにおいてDirect2Dのアクティブな実装を返します。
			取りうる値: 0, 1, 2, 3, 4 または 5 (セレクター69の値参照)。返される値はDirect2Dが利用可能かどうか、およびハードウェア、OSによってサポートされるDirec2Dの品質に基づきます。
			例えば以下のコードを実行した場合、
			`SET DATABASE PARAMETER(Direct2D Status;Direct2D Hardware)`
			`$mode:=Get database parameter(Direct2D Get active status)`
			- Windows 7以降、システムがDirect2D互換のハードウェアを検知すると、$modeに1が設定されます。そうでなければ$modeは3に設定されます (ソフトウェアコンテキスト)。
			- Windows Vistaでは、システムがDirect2D互換のハードウェアを検知すると、$modeに1が設定されます。そうでなければ$modeは0に設定されます (Direct2D無効)。
			- Windows XPでは、$modeは常に0です (Direct2D非互換)。

Diagnostic Log Recording	倍長整数	79	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **取りうる値**: 0 または 1 (0 = 記録しない, 1 = 記録する) **説明**: 4D診断ファイルの記録を開始または停止する。デフォルトの値は0で記録を行いません。 4Dは診断ファイルの中に内部的なアプリケーション処理に関連するイベントを継続的に記録することができます。このファイルに含まれる情報は4Dアプリケーション開発のために使用され、4D社の技術サポート担当により解析されます。このセレクターに1を渡すと、DatabaseName.txtという名称のファイルが自動でデータベースの**Logs**フォルダーに作成されるか、既に存在する場合は開かれます。このファイルのサイズが10MBに達するとそのファイルは閉じられ、DatabaseNameN.txtが生成されます (Nは連番)。 LOG EVENTコマンドを使用してカスタム情報をこのファイルに書き込むこともできます。

Log Command list	文字列	80	**スコープ**: 4D アプリケーション
			2セッション間で設定を保持: No
			取りうる値: 記録する4Dコマンドの番号リスト。型は文字列で各コマンド番号をセミコロンで区切ります。"all"を渡すとすべてのコマンドが記録され、"" (空の文字列) を渡すとなにも記録されません。
			説明: デバッグファイルに記録する4Dコマンドのリスト (セレクター 34, Debug Log Recording参照)。デフォルトではすべての4Dコマンドが記録されます。
			このセレクターを使用すれば、記録に残したい4Dコマンドを指定することで、デバッグファイルに保存される情報の量を制限することができます。例えば以下のようにコードを記述できます:
			`SET DATABASE PARAMETER(Log Command list;"277;341")`
			`// QUERY および QUERY SELECTION コマンドのみを記録する`

Spellchecker	倍長整数	81	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **とりうる値**: 0 (default) = OS Xのシステムのスペルチェッカー(ハンスペルは無効化されます), 1 = ハンスペルスペルチェッカー有効化 **説明**: OS X環境下においてハンスペルスペルチェッカーを有効化します。デフォルトでは、このプラットフォームではシステムのスペルチェッカーが有効化されています。例えば、クロスプラットフォームアプリケーションのインターフェースを統一するためにハンスペルを使用したいという場面があるかもしれません(Windowsでは、ハンスペルチェッカーのみが動作します)。詳細な情報に関しては、Hunspell辞書のサポートを参照して下さい。
QuickTime support	倍長整数	82	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: Yes **値**: 0 (default) = QuickTime無効, 1 = QuickTime有効 **説明**: v14以降、4DではQuickTimeコーデックはサポートされなくなりました。互換性のために、このセレクターを使用してデータベース内でQuickTimeを再有効化することができます。このオプションの変更にはデータベースの再起動が必要です。なお、将来の4DのバージョンではQuickTimeサポートは恒久的に廃止されることに注意して下さい。
JSON use local time	倍長整数	85	

例題 1

以下のメソッドを使用して4Dスケジューラーの現在の値を取得します:

```
C_LONGINT($ticksbtwcalls;$maxticks;$minticks;$lparams)
If(Application type=4D Local Mode) ` 4D local mode is used
   $lparams:=Get database parameter(4D Local Mode Scheduler)
   $ticksbtwcalls:=$lparams  & 0x00ff
   $maxticks:=($lparams>>8) & 0x00ff
   $minticks:=($lparams>>16) & 0x00ff
End if
```

例題 2

セレクタ16 (IP Address to listen) を使用して、4D WebサーバがHTTPリクエストを待ちうけるIPアドレスを取得します:

```
C_LONGINT($a;$b;$c;$d)
C_LONGINT($addr)
$addr:=Get database parameter(IP Address to listen)
$a:=($addr>>24)&0x000000ff
$b:=($addr>>16)&0x000000ff
$c:=($addr>>8)&0x000000ff
$d:=$addr&0x000000ff
```

Get last update log path

Get last update log path -> 戻り値

引数	型		説明
戻り値	テキスト	←	直近のアップデートログへのパス名

説明

Get last update log pathコマンドは、呼び出されたマシン内の最近のアップデートログファイルへの完全なパスを返します。

アップデートログはアップデーターによって生成され、4Dによって自動アップデートプロセスに使用されます。ログにはアップデートについての情報のほか、起きたエラーの情報も記録されます。

Get memory leaks

Get memory leaks
このコマンドは引数を必要としません

説明

このコマンドは内部使用のために予約されており、使用されるべきではありません。

GET MEMORY STATISTICS

GET MEMORY STATISTICS (infoType ; arrNames ; arrValues ; arrCount)

引数	型		説明
infoType	倍長整数	→	取得する情報のセレクタ
arrNames	テキスト配列	←	情報のタイトル
arrValues	実数配列	←	情報の値
arrCount	実数配列	←	関連するオブジェクトの数 (利用可能な場合)

説明

GET MEMORY STATISTICSコマンドを使用して、4Dのデータキャッシュに関連する情報を取得できます。この情報はアプリケーションの動作を検証するために使用できます。

infoType引数には取得したい情報タイプの指定する値を渡します :

- 1 = 一般的なメモリ情報。この情報は物理メモリ、仮想メモリ、空きメモリ、使用メモリ、スタックメモリ、空きスタックメモリ等のサイズ等、ランタイムエクスプローラでも見ることができるものです。
- 2 = データベースキャッシュの占有に関する統計のサマリ。

これらの値を加算して、すべての情報を一度の呼び出しで取得することもできます。コマンド実行後、リクエストされた統計は arrNames、arrValues、そして arrCount 配列に返されます。このデータの解釈に関する詳細情報は、お住まいの地域のテクニカルサポートにお問い合わせください。

GET SERIAL INFORMATION

GET SERIAL INFORMATION (key ; user ; company ; connected ; maxUser)

引数	型		説明
key	倍長整数	←	ユニークな製品キー (暗号化)
user	文字	←	登録名
company	文字	←	登録された会社名
connected	倍長整数	←	接続ユーザ数
maxUser	倍長整数	←	最大接続ユーザ数

説明

GET SERIAL INFORMATIONコマンドは、4Dのカレントバージョンのシリアル番号に関する各種情報を返します。

- key: インストールされた製品のユニークなID。固有の番号がマシンにインストールされた4Dアプリケーション（4D Server、ローカルモードの4D、4D Desktop等）に関連付けられます。もちろん、この番号は暗号化されています。
- user: インストール時に定義された、アプリケーションのユーザ名
- company: インストール時に定義された、ユーザの会社名または組織名
- connected: コマンド実行時の接続ユーザ数
- maxUsers: 同時接続最大ユーザ数

Note: 最後の2つの引数は、シングルユーザの4Dでは常に1を返し、例外としてデモバージョンでは0が返されます。

GET SERIAL INFORMATION コマンドは、4Dで採用された包括的なコンポーネント保護機構の一部です。コンポーネントの開発者は、違法コピーを防ぐため、自分の製品のコピーとインストールした所定の4Dアプリケーションとを関連付けることができます。

シリアル確認は以下のように動作します。コンポーネントを必要とするユーザは、GET SERIAL INFORMATION コマンドが返すユニークkey値をコンポーネントの開発者へ送ります。これはコンポーネントのデモバージョンに含まれる注文書を通して行われます。
コンポーネント開発者は、ユーザから送られてきた"key"値と何らかの暗号値を組み合わせて独自のシリアル番号を生成し、その番号をコンポーネントを購入したいユーザへ送ります。配付してあるコンポーネント（デモ版）にはデベロッパーより発行されたシリアル番号とGET SERIAL INFORMATIONコマンドが返す"key"値の適合性を検査する機能を盛り込んでおきます。この機能を実行し入力したシリアル番号が正しいものであればコンポーネントの機能を利用可能にします。

Note: プラグインの開発者も、この保護機構を使用することができます。詳細は4D Plugin APIリファレンスを参照してください。

Get table fragmentation

Get table fragmentation (aTable) -> 戻り値

引数	型		説明
aTable	テーブル	→	フラグメント率を取得するテーブル
戻り値	実数	↩	フラグメンテーションの割合

説明

Get table fragmentation コマンドは aTable引数で指定されたテーブルの、物理的なレコードフラグメンテーションの割合 (%) を返します。.

レコードの物理的なフラグメンテーションの割合は、データファイル中にレコードが並び順通りに格納されているかを示します。フラグメンテーション率が高くなりすぎると、テーブルに対する並び替えやシーケンシャル検索がかなり遅くなります。フラグメンテーション率0%はフラグメンテーションがないことを示します。20%を超えた場合、データファイルの圧縮をお勧めします。

例題

メンテナンスメソッドを使用して、データベース中少なくとも1つのテーブルに大きなフラグメンテーションが見つかった時に、データファイルの圧縮を要求することができます:

```
ToBeCompacted:=False
For($i ;1;Get last table number)
   If(Is table number valid($i))
      If(Get table fragmentation(Table($i)->)>20)
         ToBeCompacted:=True
      End if
   End if
End for
If(ToBeCompacted)
 // 圧縮を要求するマーカーを置く
End if
```

Is compiled mode

Is compiled mode {(*)} -> 戻り値

引数	型		説明
*	演算子	➡	ホストデータベースの情報を返す
戻り値	ブール	↩	コンパイル済み (True), インタプリタ (False)

説明

Is compiled modeは、データベースがコンパイルモード（True）、またはインタプリタモード（False）のどちらで実行されているかをテストします。

オプションの* 引数は、コンポーネントを使用するアーキテクチャで有用です。この引数は実行モードのテスト対象がホストデータベースか、コンポーネントかを指定するため に使用します。

- コマンドがコンポーネントから呼ばれた場合:
  - * 引数が渡されると、コマンドはホストデータベースの実行モードに応じてTrueまたはFalseを返します。
  - * 引数が渡されないと、コマンドはコンポーネントの実行モードに応じてTrueまたはFalseを 返します。
- コマンドがホストデータベースから呼ばれた場合、コマンドはホストデータベースの実行モードに応じてTrueまたはFalseを返します。

例題

インタプリタモードで実行している場合にだけ使用したいデバッグコードを記述するには、デバッグコードをIs compiled modeを呼び出すテストの中に記述します:

```
  ` ...
If(Not(Is compiled mode))
  ` デバッグコードをここに記述
End if
  ` ...
```

Is data file locked

Is data file locked -> 戻り値

引数	型		説明
戻り値	ブール	↩	True=ファイル/セグメントがロックされている False=ファイル/セグメントはロックされていない

説明

Is data file locked コマンドは、現在開いているデータベースのデータファイル、または少なくとも1つのセグメントがロックされている（つまり、書き込み禁止）場合にTrue（真）を返します。

例えばにおいてこのコマンドを使用すると、ロックされたデータファイルを誤ってオープンする危険性を回避することができます。

例題

このメソッドは、データファイルがロックされている場合、データベースが開かれることを回避します。

```
If(Is data file locked)
   ALERT("データファイルがロックされています。このデータベースを開くことはできません。")
   QUIT 4D
End if
```

NOTIFY RESOURCES FOLDER MODIFICATION

NOTIFY RESOURCES FOLDER MODIFICATION
このコマンドは引数を必要としません

説明

NOTIFY RESOURCES FOLDER MODIFICATION コマンドを使用して、接続されたすべての4D マシンに、**Resources**フォルダが更新された旨の通知の送信を、4D Server に強制することができます。この結果、リモートの4Dマシンはローカル**Resources**フォルダを同期できます。

このコマンドは特に、サーバ上のストアドプロシージャで**Resources**フォルダを更新した後、リモートマシンでこのフォルダの同期を管理するために使用できます。

リモートモードでの**Resources**フォルダの管理に関する詳細は、*4D Server Reference*を参照してください。

更新が行われたという情報のみが送信されます。それぞれのリモートマシンは、ローカルの環境設定に従い処理を行います。オプションは以下のとおりです:

- セッション中ローカルの**Resources**フォルダを同期しない。
- セッション中にローカルの**Resources**フォルダを自動で同期する
- 警告を表示し、同期を行うかユーザに選択させる

設定は以下のいずれかの方法で行います:

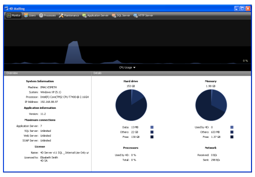
- データベース設定の"Resources"フォルダをセッション中に更新を使用してデータベース全体を対象に設定します。この場合、すべてのマシンに設定が適用されます。
- SET DATABASE PARAMETERコマンドを使用して各リモートマシンごとに設定します (Auto Synchro Resources Folder セレクター)。この場合、この設定はデータベース設定を上書きし、セッション中そのリモートマシンにのみ適用されます。

OPEN ADMINISTRATION WINDOW

OPEN ADMINISTRATION WINDOW
このコマンドは引数を必要としません

説明

OPEN ADMINISTRATION WINDOWコマンドは、コマンドを実行したマシン上でサーバ管理ウィンドウを開きます。4D Serverの管理ウィンドウで現在のパラメタを表示させたり、さまざまなメンテナンス操作を行ったりできます (4D Server Reference Guide参照)。4D Server のバージョン11から、このウィンドウをクライアントマシン上で表示できるようになりました:



System Information

Machine: IMAC-ESMITH
System: Windows XP (5.1)
Processor: Intel(R) Core(TM)2 CPU T7400 @ 2.16GH
IP Address: 192.168.88.97

Application information

Version: 11.2

Maximum connections

Application Server: 7
SQL Server: Unlimited
Web Server: Unlimited
SOAP Server: Unlimited

License

Name: 4D Server v11 SQL _ Internal Use Only ur
Licenced to: Elizabeth Smith
4D SA

Hard drive

153 GB

Data: 13 MB
Others: 22 GB
Free: 130 GB

Memory

1.98 GB

Used by 4D: 0
Others: 633 MB
Free: 1.37 GB

Processors

Used by 4D: 0 %
Total: 0 %

Network

Received: 0 B/s
Sent: 298 B/s

このコマンドは、4D Serverに接続した4Dアプリケーションまたは4D Serverから呼び出さなければなりません。以下の場合、コマンドは何も行いません:

- ローカルモードの4Dアプリケーションから呼び出された場合。
- Designer でもAdministrator でもないユーザが実行した場合 (この場合エラー-9991 が生成されます。データベースエンジンエラー (-10600 -> 4004)参照)。

例題

この例を管理ボタンに割り当てることができます:

```
If(Application type=4D local mode)
   OPEN SECURITY CENTER
 // ...
End if
If(Application type=4D remote mode)
   OPEN ADMINISTRATION WINDOW
 // ...
End if
If(Application type=4D Server)
 // ...
   OPEN SECURITY CENTER
End if
```

システム変数およびセット

コマンドが正しく実行されるとOKシステム変数に1が設定されます。そうでなければ0が設定されます。

OPEN DATA FILE

OPEN DATA FILE (accessPath)

引数	型		説明
accessPath	文字	→	開くデータファイルの名前または完全アクセスパス

説明

OPEN DATA FILEコマンドは、4Dアプリケーションによって開かれたデータファイルをオンザフライで変更することを可能にします。

accessPath 引数には、開こうとするデータファイル (".4DD"拡張子を持つファイル) の名前または完全なアクセスパスを渡します。ファイル名だけを渡す場合、データファイルはデータベースのストラクチャファイルと同じ階層に配置されていなければなりません。

アクセスパスが有効なデータファイルを指している場合、4Dは現在使用しているデータベースを終了し、指定されたデータファイルを使って再度開きます。シングルユーザーモードではOn Exitデータベースメソッドと On Startupデータベースメソッドが続いて呼び出されます。

警告： このコマンドを使用すると、指定されたデータファイルで再びデータベースが開かれる前に、アプリケーションは一度終了します。このためOn Startupデータベースメソッドやこのデータベースメソッドから呼び出されるメソッド内で、このコマンドを使用することはできません。

このコマンドは非同期的に実行されます。つまり、コマンド呼び出しの後、4Dはメソッドの残りの部分を続けて実行します。この後、アプリケーションは**ファイル**メニューの**終了**コマンドが選択された場合と同様の処理を行います。表示されているダイアログボックスはキャンセルされ、実行中のプロセスは10秒間の猶予の後に打ち切られます。

処理を開始する前に、コマンドは指定されたデータファイルが有効かどうかを検証します。また、そのファイルが既に開かれている場合、コマンドはファイルが現在使用中のストラクチャファイルに対応するかどうかを確認します。

accessPathに空の文字列を渡した場合、このコマンドはデータファイルを変更せずにデータベースを再度開きます。

4D Server: 4D v13より、このコマンドを4D Serverで実行できるようになりました。このコンテキストで、このコマンドは指定されたファイルを開く前に、サーバー上で内部的にQUIT 4Dを呼び出します。結果各接続クライアント上にはサーバーが終了中である旨を通知するダイアログが表示されます。

OPEN SECURITY CENTER

OPEN SECURITY CENTER
このコマンドは引数を必要としません

説明

OPEN SECURITY CENTER コマンドは、Maintenance and Security Center (MSC) ウィンドウを表示します。

カレントユーザのアクセス権に基づき、ウィンドウ中の機能は利用不可になります。

OPEN SETTINGS WINDOW

OPEN SETTINGS WINDOW (selector {; access}{; settingsType})

引数	型		説明
selector	文字	→	環境設定やデータベース設定ダイアログボックス中のテーマやページ、またはパラメーターグループを指定するキー
access	ブール	→	True=ダイアログボックスの他のページをロックする Falseまたは省略=ダイアログの他のページもアクティブにする
settingsType	倍長整数	→	0または省略時 = ストラクチャー設定、1 = ユーザー設定

説明

OPEN SETTINGS WINDOWコマンドは、カレントの4Dアプリケーションの環境設定ダイアログボックスまたはカレント4Dアプリケーションのデータベース設定を表示し、selectorキーで指定されたにパラメターやページを表示します。

selector 引数には、開くダイアログボックスとそのページを示す"キー"を渡さなくてはなりません。キーは以下のような構造になります: /Dialog{/Page{/Parameters}}。*Dialog* は表示するダイアログで、"4D" (環境設定) または"Database" (データベース設定) を指定できます。例えばデータベース設定のコンパイラページを指定するには、selector に"/Database/Compiler"を指定します。使用可能なキーのリストは後に示します。selectorにスラッシュ ("/") のみを渡すと、コマンドはデータベース設定の最初のページを表示します。

access引数を使用して他のページをロックすることで、環境設定やデータベース設定ダイアログボックス内でのユーザーアクションをコントロールできます。特に、ユーザーに対し特定のパラメタの変更を許可し、他の部分の変更は許可したくないという場合に使用します。この場合、access引数にTrueを渡すとselector引数で指定したページのみがアクティブになり変更可能となります。他のすべてのページへのアクセスはロックされます (ナビゲーションバーのボタンをクリックしても効果はありません)。access引数にFalseを渡すか省略すると、ダイアログボックスのすべてのページがアクセス可能になります。

settingsType 引数はユーザー設定モードに設定されたデータベースでのみ有効です。このモードではカスタマイズされたユーザー設定が外部ファイルとして生成され、標準設定の代わりに使用されます。詳細はデザインリファレンスマニュアルを参照してください。このコンテキストでは、この引数を使用してストラクチャー設定またはユーザー設定いずれのダイアログボックスにアクセスするか指定できます。4D Environmentテーマの以下の定数を使用できます:

定数	型	値	コメント
Structure Settings	倍長整数	0	ストラクチャー設定を使用 (引数が省略された際のデフォルト)。このモードでは使用されるselectorの値は標準モードと同じです。
User Settings	倍長整数	1	ユーザー設定を使用。このモードでは特定のキーのみをselector引数で使用できます。

無効なキーを渡すと、データベース設定の最初のページが表示されます。

キーのパス (標準モード)

以下は標準モードのselector引数 (ストラクチャー設定) で使用可能なキーです:

/4D
/4D/General
/4D/Structure
/4D/Form editor
/4D/Method editor
/4D/Client-Server
/4D/Shortcuts
/Database
/Database/General
/Database/Mover
/Database/Interface
/Database/Interface/Developper
/Database/Interface/User
/Database/Interface/Shortcuts
/Database/Compiler
/Database/Database
/Database/Database/Data storage
/Database/Database/Memory and cpu
/Database/Database/International
/Database/Backup
/Database/Backup/Scheduler
/Database/Backup/Configuration
/Database/Backup/Backup and restore
/Database/Client-Server
/Database/Client-Server/Network
/Database/Client-Server/IP configuration
/Database/Web
/Database/Web/Config
/Database/Web/Options 1
/Database/Web/Options 2
/Database/Web/Log format
/Database/Web/Log scheduler
/Database/Web/Webservices
/Database/SQL
/Database/php
/Database/Compatibility
/Database/Security

互換性に関する注意: このコマンドは以前のバージョンのキーでも動作します。対応は自動で4Dが行います。しかしながら以前の呼び出しを、上記のキーで置き換えることを推奨します。

Path keys (User settings mode)

以下はユーザー設定モードのselector引数で使用できるキーです:

/Database
/Database/Interface
/Database/Database/Memory and cpu
/Database/Client-Server
/Database/Client-Server/Network
/Database/Client-Server/IP configuration
/Database/Web
/Database/Web/Config
/Database/Web/Options 1
/Database/Web/Options 2
/Database/Web/Log format
/Database/Web/Log scheduler
/Database/Web/Webservices
/Database/SQL
/Database/php

例題 1

4D環境設定の"メソッド"ページを開きます:

```
 OPEN SETTINGS WINDOW("/4D/Method editor")
```

例題 2

データベース設定の"ショートカット"パラメタを開き、他の設定はロックします:

```
 OPEN SETTINGS WINDOW("/Database/Interface/Shortcuts";True)
```

例題 3

環境設定の最初のページを開きます:

```
 OPEN SETTINGS WINDOW("/")
```

例題 4

ユーザー設定モードでデータベース設定のインターフェースページにアクセスする:

```
 OPEN SETTINGS WINDOW("/Database/Interface";1)
```

システム変数およびセット

環境設定/データベース設定ダイアログが受け入れられるとOKシステム変数に1が、キャンセルされると0が設定されます。

PLUGIN LIST

PLUGIN LIST (numbersArray ; namesArray)

引数	型		説明
numbersArray	倍長整数配列	⇐	プラグイン番号
namesArray	文字配列	⇐	プラグイン名

説明

PLUGIN LIST コマンドはnumbersArray配列とnamesArray配列に、4Dアプリケーションにロードされ、利用可能なプラグインの番号と名前を返します。これら2つの配列は、コマンドによりサイズが調整され、同期されます。

注: numbersArray配列に返された値と、Is License Availableテーマの定数の値とを比較できます。

PLUGIN LISTは、4Dに統合されたプラグイン (例: 4D Chart) やサードパーティのプラグインも対象とします。

QUIT 4D

QUIT 4D {(time)}

引数	型		説明
time	倍長整数	⇒	サーバ終了までの時間 (秒)

説明

QUIT 4Dコマンドは、カレントの4Dアプリケーションを終了してデスクトップに戻ります。

4Dまたは4D Serverどちらでコマンドが実行されたにより、このコマンドは異なる処理を行います。

ローカルモードおよびリモートモードの4D

QUIT 4Dを呼び出した後、カレントのプロセスは実行を停止し、4Dは以下のように振舞います:

- On Exitデータベースメソッドがある場合、4Dはローカルプロセスを新しく作成し、その内でこのデータベースメソッドの実行を開始します。例えばこのデータベースメソッドを使用して、開発者は他のプロセスに、プロセス間通信を駆使し、データ入力やOn Startupデータベースメソッドで開始された処理 (他のデータサーバーへの接続等) を停止するよう、通知できます。4Dは、いずれ終了してしまうことを覚えておいてください。On Exitデータベースメソッドは、必要なすべてのクリーンアップ操作や終了を実行しますが、終了を拒否することはできないため、ある時点でデータベースは終了します。
- On Exitデータベースメソッドがない場合、4Dは実行プロセスそれぞれを区別せずに1つずつアボートします。

ユーザがデータ入力を実行している場合には、レコードはキャンセルされ、保存されません。
現在開いているウィンドウ上で行われたデータ入力の変更内容をユーザに保存させたい場合は、プロセス間通信を使って、データベースが終了されようとしていることを他のすべてのユーザプロセスに通知することができます。これを実行するには、以下の2つの方法があります:

- この処理を、**QUIT 4D**を呼び出す前に、カレントのプロセスで行う。
- この処理を、On Exitデータベースメソッド内で行う。

3番目の方法もあります。**QUIT 4D**コマンドを呼び出す前に、ウィンドウで行われた処理が登録を必要とするかどうかをチェックします。このケースの場合は、ユーザにこのウインドウの内容をを保存するか、または取り消すかを尋ね、それから再度、「終了」を選択するか尋ねます。しかし、ユーザインタフェースの観点から見ると、最初の2つの方法を使用するのをお勧めします。

注: 4Dでは、time引数を使用できません。

4D Server (ストアドプロシージャ)

QUIT 4Dコマンドを、サーバマシン上のストアドプロシージャで実行できます。その場合、オプションの引数timeを受け付けます。
time引数により、アプリケーションが実際に終了するまでの4D Serverのタイムアウトを設定し、クライアントマシンが接続を切るための時間を与えることができます。timeには分単位の値を渡します。この引数は、サーバマシン上で実行された際にのみ考慮されます。4Dでは、この引数は無視されます。
time引数を渡さない場合、終了する前に4D Serverはすべてのクライアントマシンが接続を切るまで待機します。

4Dとは異なり、4D Serverによる**QUIT 4D**の処理は非同期的に行われます。つまり、このコマンドの実行後、コマンドの呼び出しを行ったメソッドが中断されることはありません。

On Server Shutdownデータベースメソッドが存在する場合、渡された引数に基づき、time引数で設定された遅延後またはすべてのクライアントの接続が解除された後、このメソッドが実行されます。

ストアドプロシージャ内で使用された場合の**QUIT 4D**コマンドの動作は、4D Serverの「ファイル」メニューから「終了」コマンドを選択した場合と同じです。すなわち、各クライアントマシンにダイアログボックスが表示され、サーバが終了することを通知します。

例題

以下のプロジェクトメソッドは、ファイルメニューの終了コマンドに割り当てられます。

```
  ` M_FILE_QUIT プロジェクトメソッド
CONFIRM("本当に終了しますか?")
If(OK=1)
   QUIT 4D
End if
```

RESTART 4D

RESTART 4D {(time {; message})}

引数	型		説明
time	倍長整数	⇒	4Dが再起動するまでの猶予時間(秒)
message	文字	⇒	クライアントマシンに表示するテキスト

説明

RESTART 4Dコマンドは、カレントの4D組み込みアプリケーションを再起動します。

このコマンドは組み込みアプリ(クライアント/サーバーアプリまたはシングルユーザー向けアプリ)に対しての使用を目的としており、SET UPDATE FOLDER コマンドとの併用する必要があります。これにより、自動アップデートプロセスが発動し、**RESTART 4D**コマンドによってアプリケーションが再起動し、SET UPDATE FOLDERによって指定されたアプリケーションの新バージョンがカレントのバージョンを置き換えます。データファイルへのパス名は自動的に保存され、使用されます。

Note: カレントセッションにおいてSET UPDATE FOLDERコマンドによってアップデート情報が何も指定されていなかった場合、このコマンドは単に組み込みアプリケーションを同じデータファイルで再起動します。

RESTART 4D はこの用法以外での使用はできません。特に、組み込みがされていない4Dデータベースに対しては効きません。

time 引数を使用して、クライアントマシンが接続を切断するために再起動までの時間を指定することが出来ます。time 引数には秒単位の値を渡さなければなりません。この引数を省略した場合、サーバーアプリケーションは10分を上限として全てのクライアントアプリケーションが接続を切断するのを待ちます。それ以降は全てのクライアントアプリケーションは自動的に切断されます。

Note: time 引数はサーバーアプリケーションにおいてのみ考慮されます(シングルユーザー向けアプリでは無視されます)。

任意の message 引数は接続済みのクライアントアプリケーションに対してのカスタムのメッセージを表示します。

コマンドが正常に実行されれば、OKシステム変数は1に設定されます。それ以外の場合は0に設定され、アプリケーションは再起動します。このコマンドによって生成されるエラーは全て ON ERR CALL コマンドによって実装されたメソッドによって割り込み可能です。

SET DATABASE LOCALIZATION

SET DATABASE LOCALIZATION (languageCode {; *})

引数	型		説明
languageCode	テキスト	⇒	言語セレクタ
*	演算子	⇒	コマンドのスコープ

説明

SET DATABASE LOCALIZATIONコマンドを使用して、カレントセッションのデータベースカレント言語を変更できます。

データベースのカレント言語は、アプリケーションのローカライズされた要素 (テキストおよびピクチャ) をプログラムが検索する場所である.lprojフォルダを指定します。デフォルトで4Dは、Resourcesの内容およびシステム環境に基づき自動でカレント言語を決定します (Get database localization コマンドの説明参照)。**SET DATABASE LOCALIZATION**を使用して、デフォルトのカレント言語を変更できます。

コマンドは既にロードされたフォームの言語を変更しません。コマンドが呼び出された後に表示された要素のみが、新しい設定の効果を得ます。

languageにはアプリケーションで使用する、RFC 3066、ISO639 そして ISO3166標準で規定された言語コードを渡します。例えば日本語であれば"ja"、フランス語なら"fr"、アメリカ英語なら"en-us"を渡します。この標準に関する詳細や、渡すことが可能な値については*Design Reference*マニュアルの付録 C: XLIFFアーキテクチャーを参照してください。

デフォルトで、コマンドは開かれたすべてのデータベースとコンポーネント、およびすべてのプロセスに適用されます。オプションの* 引数が渡されると、このコマンドを呼び出したデータベースにのみ適用されます。この引数は特に、データベースとコンポーネントで別々に言語を指定するために使用されます。

コマンドが正しく実行されるとOKシステム変数に1が設定されます。そうでなければ0が設定されます。

Note: RFCに従い、コマンドは言語コードと地域コードを分けるために"-" (ハイフン) を使用します (例えば"fr-ca"や"en-us")。他方、**Resources**フォルダ内の"lproj"フォルダは"_" (アンダースコア)を使用します (例えば"fr_ca.lproj"や"en_us.lproj")。

4D Server: 4D Serverでは、コマンドを呼び出したリモートマシン上に存在する言語を利用できます。そのため、**Resources**フォルダが 同期されているか確かめなければなりません。

例題 1

日本語をインタフェース言語として設定する場合:

```
 SET DATABASE LOCALIZATION("ja")
```

例題 2

アプリケーションで文字列参照":xliff:shopping"が使用されていて、XLIFFファイルには以下のような情報が含まれています:

- JA folder:

```
  <trans-unit id="15" resname="Shopping">
<source>Shopping</source>
<target>ショッピング</target>
</trans-unit>
```

- FR folder:

```
 <trans-unit id="15" resname="Shopping">
<source>Shopping</source>
<target>Faire les courses</target>
</trans-unit>
```

```
 SET DATABASE LOCALIZATION("fr")
  // 文字列":xliff:shopping"は"Faire les courses"を表示する
 SET DATABASE LOCALIZATION("ja")
  // 文字列":xliff:shopping"は"ショッピング"を表示する
```

SET DATABASE PARAMETER

SET DATABASE PARAMETER ({aTable ;} selector ; value)

引数	型		説明
aTable	テーブル	→	パラメータをセットするテーブル、または 省略時、デフォルトテーブル
selector	倍長整数	→	変更を行うデータベースパラメタのコード
value	実数, 文字	→	パラメタの値

説明

SET DATABASE PARAMETERコマンドを使用して、4Dデータベース内部の様々なパラメーターを変更することができます。

selectorには、変更するデータベースのパラメーターを指定します。4DはDatabase Parametersテーマに定義済の下記のような定数があります。次の表は、各定数とその有効範囲、またその設定が2つの異なるセッションで保持されるかを示しています:

定数	型	値	コメント
Direct2D Disabled	倍長整数	0	セレクター69 (Direct2D Status) 参照。
Direct2D Hardware	倍長整数	1	セレクター69 (Direct2D Status) 参照。
Seq Order Ratio	倍長整数	1	*** このセレクターは廃止されました ***
Direct2D Hardware SVG and Editors	倍長整数	2	セレクター69 (Direct2D Status) 参照。
Seq Access Optimization	倍長整数	2	*** このセレクターは廃止されました ***
Direct2D Software	倍長整数	3	セレクター69 (Direct2D Status) 参照。
Seq Distinct Values Ratio	倍長整数	3	*** このセレクターは廃止されました ***
Direct2D Software SVG and Editors	倍長整数	4	セレクター69 (Direct2D Status) 参照。
Index Compacting	倍長整数	4	*** このセレクターは廃止されました ***
Direct2D Mixed	倍長整数	5	セレクター69 (Direct2D Status) 参照。
Seq Query Select Ratio	倍長整数	5	*** このセレクターは廃止されました ***
Minimum Web Process	倍長整数	6	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **とりうる値**: 0 -> 32 767 **説明**: ローカルモードの4Dならびに4D Serverを使用した場合に、非コンテキストモードで保持するWebプロセスの最小数。デフォルト値は0（下記参照）。
Maximum Web Process	倍長整数	7	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **とりうる値**: 0 -> 32 767 **説明**: ローカルモードの4Dならびに4D Serverを使用した場合に、非コンテキストモードで保持するWebプロセスの最大数。デフォルト値は10。 非コンテキストモードでのWebサーバの反応を良くするため、4Dは5秒間Webプロセスを遅延させ、将来やってくるかもしれないHTTPクエリを処理する際、それを再利用します。パフォーマンスの観点では、クエリごとに新しいプロセスを作成するよりも、再利用したほうが実際に有利です。Webプロセスが再利用されると、さらに5秒間遅延されます。Webプロセスが最大数に達するとプロセスがアボートされます。5秒の遅延以内にWebプロセスがクエリを受け取らない場合、Webプロセスの最小数を下回らなければ、プロセスはアボートされます。最小数未満になる場合、プロセスは再度遅延されます。 これらの引数は、リクエスト数や利用可能なメモリ、その他のパラメタに応じて、Webサーバの動作を調整できるようにするものです。
Web Conversion Mode	倍長整数	8	**スコープ**: カレントプロセス **2セッション間で設定を保持**: No **とりうる値**: 0, 1, 2 または 3 • (デフォルトモード) ブラウザが対応する場合、HTML 4.0フォーマットに変換。そうでなければHTML 3.2フォーマットと配列を使用。 • 1 = 6.0.x変換モード • 2 = 6.5変換モード • 3 = HTML 4.0フォーマット + CSS-P (バージョン6.5.2より) に変換。 **説明**: ローカルモードの4Dならびに4D Serverで使用するWeb用の4Dフォーム変換モード。デフォルトで、4D WebサーバはCSS1を使用し、4Dで表示される4Dフォームと同様のHTMLページを生成します。この機能では、バージョン6.7より前の4Dで作成されたデータベースに関しては、フォームが正しく変換されない可能性があります。このため、フォーム変換モードを設定する必要があるかもしれません。 このモードは、SET DATABASE PARAMETERが呼び出されたプロセス（Webコンテキスト）に対してのみ設定されます。このコマンドは、メソッド内で呼び出してデータベースのすべてのフォームを確実に統一したり、あるいは特定のフォームを表示する前にのみ呼び出すこともできます。このコマンドは、コンテストモード、またはWebプロセス以外の場所から呼び出すと、何も行いません。
Database Cache Size	倍長整数	9	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: - **説明**: カレントのデータベースメモリキャッシュサイズを取得するために使用します。戻り値はバイト単位です。 最大キャッシュサイズは環境設定の"データベース/データ管理"ページで設定できます。しかしデータベースキャッシュに実際に割り当てられる値は設定とシステムの現在のリソースに基づきます。このセレクタを使用すれば4Dによりデータベースに割り当てられた実際のメモリサイズを取得できます。 **警告**: ランゲージを使用してデータベースキャッシュメモリサイズを設定することはできません。言い換えればDatabase Cache SizeをSET DATABASE PARAMETERコマンドで使用することはできません。
4D Local Mode Scheduler	倍長整数	10	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: Yes **説明**: セレクタ12参照
4D Server Scheduler	倍長整数	11	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: Yes **説明**: セレクタ12参照
4D Remote Mode Scheduler	倍長整数	12	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ10, 11 および 12に対し、value引数は16進数、0x00aabbccの形式で表わされます。詳細は次の通りです: *aa* = システムへのコール毎の最小tick数（0～100） *bb* = システムへのコール毎の最大tick数（0～100） *cc* = システムへのコール間のtick数（0～20） これらの値が範囲外のとき、4Dはその値を最大数に設定します。value引数には、次の定義済標準値のうちいずれかを渡すことができます: • value = -1: 4Dに最高優先度を割り当てる • value = -2: 4Dに平均的な優先度を割り当てる • value = -3: 4Dに最低優先度を割り当てる **説明**: この引数を使用して、4Dシステム内部コールをダイナミックに設定することができます。selectorに応じて、スケジューラの値は次のアプリケーションのために設定されます。 • シングルユーザの4Dから呼び出された場合、ローカルモードの4D (selector=10). • 4D Serverから呼び出された場合、4D Server (selector=11). • 4DServerに接続した4Dから呼び出された場合、リモートモードの4D (selector=12). **Note:** セレクタ=12(4D Remote Mode Scheduler)は、SET DATABASE PARAMETERコマンドがサーバーマシン上で実行されるか、クライアントマシン上で実行されるかにより異なります: • コマンドがサーバーマシン上で実行される場合、新しい値はコマンド実行後にサーバに接続する全てのクライアントマシンに適用されます。 • コマンドがクライアントマシン上で実行される場合、新しい値はそのクライアントマシンで直ちに有効となり、またコマンド実行後にサーバに接続するすべてのクライアントマシンに対し適用されます。 この動作を使用することで、クライアントマシン毎に異なる特性をダイナミックに扱うことが可能です。これはコマンドの実行がまず、そのクライアントマシンの設定に影響し、その後にサーバに接続するすべてのクライアントマシンへの設定値として使用されるという動作に基づきます。 この動作は、4Dの6.8.6, 2003.3と4D2004以降のバージョンで効果があります。 **警告**: これらのセレクタを不適切に使用すると、アプリケーションパフォーマンスが深刻に低下する原因となります。これらの要素に対する完全な知識がある場合のみ変更を行うことをお勧めします。
4D Server Timeout	倍長整数	13	**スコープ**: value が正数なら4Dアプリケーション **2セッション間で設定を保持**: value が正数ならYes **とりうる値**: 0 -> 32 767 **説明**: この引数を使用して、4D Serverのタイムアウトの値を変更することができます。4D Serverのタイムアウトのデフォルト値は、サーバ側の「環境設定」ダイアログボックスの「クライアント-サーバ/接続設定」ページで定義します。 セレクタ4D Server Timeoutにより、対応する引数valueの新しいタイムアウト（分単位で指定）を設定できます。この機能は、クライアント側でCPUを占有する、時間がかかる処理を実行する前に、タイムアウト設定を長くしたい場合は特に便利です。例えば、膨大なページの印刷などは、予期しないタイムアウトになる可能性があります。 2種類のオプションがあります: • value引数に**正数**を渡すと、グローバルかつ永続的なタイムアウトが設定されます。この新しい値はすべてのプロセスに対して適用され、4Dアプリケーションの環境設定に保存されます（環境設定ダイアログボックスで変更した場合と同じ）。 • value引数に**負数**を渡すと、ローカルで一時的なタイムアウトが設定されます。この新しい値は呼び出し元のプロセスに対してのみ適用され（他のプロセスではデフォルトの値を維持）、例えば処理の終了時のように、クライアントが動作していることを示す信号をサーバが受信すると即座に、デフォルト値へリセットされます。このオプションは、4Dプラグインにより開始された時間のかかる処理を管理する際に便利です。 "タイムアウトしない"オプションを設定するには、valueに0 を渡します。
4D Remote Mode Timeout	倍長整数	14	**スコープ**: value が正数の場合4D アプリケーション **2セッション間で設定を保持**: value が正数の場合Yes **説明**: この引数を使用して、4D Serverに接続したリモートの4Dマシンが参照するタイムアウト値を設定できます。リモートモードの4Dが使用するデフォルトのタイムアウト値は、リモートマシン上の環境設定の"クライアント-サーバ/設定"ページで設定されます。 このセレクタに関する詳細は4D Server Timeout (13) の説明を参照してください。4D Remote Mode Timeout セレクタは非常に特殊な状況において使用されます。
Port ID	倍長整数	15	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: No **説明**: この引数を使用して、ローカルモードの4Dおよび4D Serverによる4D Webサーバが使用するTCPポートをオンザフライで変更することができます。デフォルト値は80で、この値は環境設定ダイアログボックスの"Web/設定"ページで設定できます。TCP Port Numbersテーマの定数をvalue引数に使用できます。 Port IDセレクタは、コンパイルして4D Desktopを組み込んだ4D Webサーバで役立ちます（この場合、デザインモードへのアクセス手段がありません）。TCPポートIDに関する詳細はWebサーバー設定を参照してください。
IP Address to listen	倍長整数	16	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Character set	倍長整数	17	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Max Concurrent Web Processes	倍長整数	18	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Client Minimum Web Process	倍長整数	19	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ6参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Maximum Web Process	倍長整数	20	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ7参照 **Description**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Max Web requests size	倍長整数	21	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ27参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Port ID	倍長整数	22	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ15参照 **Description**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client IP Address to listen	倍長整数	23	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ16参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Character set	倍長整数	24	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ17参照 **Description**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Max Concurrent Web Proc	倍長整数	25	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ18参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Cache writing mode	倍長整数	26	*** このセレクターは廃止されました ***
Maximum Web requests size	倍長整数	27	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
4D Server Log Recording	倍長整数	28	**Scope**: 4D Server, 4D リモート **2セッション間で設定を保持**: No **とりうる値**: 0 または 1 から X (0 = 記録しない, 1 から X = ファイル名に付加するシーケンス値)。 **説明**: 4D Serverが受け取る標準的なリクエスト（Webリクエストを除く）の記録を開始または停止します。デフォルト値は0（リクエストを記録しない）です。 サーバマシンが受信した各リクエストをログファイルに記録するよう、4D Serverに指示することができます。このメカニズムが有効になると、データベースストラクチャと同じ階層にログファイルが作成されます。ファイルには"4DRequestsLogX"（Xはログのシーケンシャル番号）の名前が付けられます。ファイルサイズが10MBに達するとそのファイルは閉じられ、インクリメントしたシーケンス番号を使用して新しいファイルが作成されます。もし同じ名前のファイルが存在する場合は置き換えられます。シーケンシャル番号の開始番号を、value引数を使用して指定できます。 このテキストファイルにはそれぞれのリクエストに関する様々な情報（時間、プロセス番号、ユーザ、リクエストサイズ、処理時間など）がシンプルな表形式のフォーマットで保存されます。この情報はアプリケーションの調整を行う場合や統計の目的で利用する場合に特に役立ちます。この情報を、例えばスプレッドシートソフトウェアに読み込んで処理することもできます。 **Note:** マニュアル操作により、ログの記録をOn/Offすることが出来ます。Windowsでは**Ctrl+Alt+L**、MacOSでは**Command+Option+L**のショートカットです。
Web Log Recording	倍長整数	29	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Client Web Log Recording	倍長整数	30	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: 0 = 記録しない (デフォルト), 1 = CLFフォーマットで記録, 2 = DLFフォーマットで記録, 3 = ELFフォーマットで記録, 4 = WLFフォーマットで記録 **説明**: すべてのクライアントマシンのWebサーバが受け取るWebリクエストの記録を開始または停止します。デフォルト値は0（リクエストを記録しない）です。 このセレクタの動作はセレクタ29と同じです。しかし対象はWebサーバとして使用されるすべての4Dクライアントマシンです。この場合、"logweb.txt"ファイルは4Dクライアントデータベースフォルダ (キャッシュフォルダ) のLogsサブフォルダに作成されます。特定のクライアントマシンにのみ設定を行いたい場合、4Dクライアントの環境設定を使用します。
Table Sequence Number	倍長整数	31	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: Yes **とりうる値**: 任意の倍長整数値 **説明**: このセレクタは、引数に渡したテーブルのレコードの、カレントのユニーク番号を取得あるいは設定するために使用します。"カレントの数値"とは"最後に使用された数値"を意味します。SET DATABASE PARAMETERコマンドを使用してこの値を変更すると、渡された値+1の番号を使用して次のレコードが作成されます。この新しい番号は、Sequence numberコマンドによって返される、さらにはストラクチャエディタやSQLで自動インクリメントが設定されたフィールドに返される番号です。 デフォルトで、この固有の番号は4Dが設定し、レコードの作成順に対応します。詳細はSequence numberコマンドのドキュメントを参照してください。
_O_Real display precision	倍長整数	32	*** このセレクターは廃止されました ***
TCP_NODELAY	倍長整数	33	*** このセレクターは廃止されました ***
Debug log recording	倍長整数	34	
Client Server Port ID	倍長整数	35	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0～65535 **説明**: 4D Server が（4D Client に対して）データベースを公開するために使用されるTCPポート番号をプログラムで変更するために使用します。デフォルト値は19813 です。 この値を変更すれば、TCPプロトコルを使用して、複数の4D クライアント/サーバアプリケーションを同じマシンで同時に使用することができます。その場合、それぞれのアプリケーションごとに異なるポート番号を設定します。 公開ポート番号は、ストラクチャファイルに記録されています。ローカルモードの4Dで設定することもできますが、クライアント/サーバ環境でのみ考慮されます。 値を変更した場合、新しい設定が有効になるためには、アプリケーションを再起動する必要があります。
Invert Objects	倍長整数	37	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0, 1 または 2 (0 = モードは無効, 1 = 自動モード, 2 = モードが有効) **説明**: Right-to-left 言語のWindowsでデータベースが表示されるとき、アプリケーションモードでフォームやオブジェクト、メニューバーなどを反転させるために使用される、"オブジェクト反転"モードを設定します。このモードはアプリケーション環境設定のデータベース/インターナショナルページで変更できます。 • 0 に設定した場合、システム設定に関係なく、モードは無効です（環境設定でいいえにするのと同じ）。 • 1 に設定した場合、システム設定に応じ、モードが有効または無効になります（環境設定を自動にするのと同じ）。 • 2 に設定した場合、システム設定に関係なく、モードは有効です（環境設定をはいにするのと同じ）。 詳細は4DのDesign Referenceマニュアルを参照してください。
HTTPS Port ID	倍長整数	39	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Client HTTPS Port ID	倍長整数	40	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: 0 ～ 65535 **説明**: このセレクタは、クライアントマシンのWebサーバがSSLによるセキュアな接続（HTTPSプロトコル）で使用するTCPポート番号を、プログラムで変更するために使用できます。デフォルトの値は443（標準ポート番号）です。 このセレクタの動作はセレクタ39と同じですが、Web サーバとして使用されるすべてのクライアントマシンに適用されます。特定のクライアントマシンの設定だけを変更するのであれば、4Dリモートの環境設定ダイアログ画面を使用してください。
Unicode mode	倍長整数	41	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0 (互換モード) または 1 (Unicodeモード) **説明**: カレントデータベースの文字セットに関連する動作モード。4DはUnicode文字セットをサポートしますが、(Mac ASCII文字セットに基づく)"互換"モードで動作させることができます。デフォルトで、変換されたデータベースは互換モード (0) で、バージョン11以降で作成されたデータベースはUnicodeモードで実行されます。実行モードは環境設定のオプションでコントロールでき、またこのセレクタを使用して読みだしたり、(テスト目的で) 変更したりできます。このオプションを変更した場合、それを有効にするにはデータベースを再起動しなければなりません。コンポーネント内部ではこの値を変更できないことに留意してください。読み出しのみが可能です。
Temporary memory size	倍長整数	42	*** このセレクターは廃止されました ***
SQL Autocommit	倍長整数	43	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0 (無効) または 1 (有効) **説明**: SQLの自動コミットモードを有効または無効にするために使用します。デフォルトは 0 (無効モード) です。 自動コミットモードは、データベースの参照整合性を強化するために使用されます。このモードが有効の時、すべてのSELECT, INSERT, UPDATE そして DELETE (SIUD) クエリは、これらがトランザクション内で実行されていない場合、自動でアドホックなトランザクションに含められます。このモードはデータベースの環境設定でも設定できます。
SQL Engine Case Sensitivity	倍長整数	44	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0 (大文字小文字を考慮しない) または 1 (考慮する) **説明**: SQLエンジンが文字列比較を行う際に、大文字と小文字の違いを考慮させるかどうかを設定します。 デフォルトで値は1 (大文字小文字を考慮する) です。SQLエンジンは文字列比較 (並び替えやクエリ) の際に大文字と小文字を異なる文字として扱います。例えば"ABC"= "ABC"ですが"ABC" # "Abc"です。SQLエンジンと4Dエンジンの動作をそろえたいなど特定の場合では、大文字と小文字を区別しない文字列比較 ("ABC"="Abc") を使用できます。 このオプションはアプリケーション環境設定の SQL/設定ページで設定できます。
Client Log Recording	倍長整数	45	**スコープ**: リモート4Dマシン **2セッション間で設定を保持**: No **とりうる値**: 0 または 1 から X (0 = 記録しない, 1 から X = ファイル名に付加するシーケンス値)。 **説明**: コマンドを実行した4Dクライアントマシンが実行した標準的なリクエスト（Webリクエストを除く）の記録を開始または停止します。デフォルト値は0（リクエストを記録しない）です。 クライアントマシンが実行したリクエストをログファイルに記録するよう、4Dに指示することができます。このメカニズムが有効になると、クライアントマシンのデータベースのローカルフォルダ内、Logsサブフォルダに2つのログファイルが作成されます。ファイルには"4DRequestsLog_X"と"4DRequestsLog_ProcessInfo_X"（Xはログのシーケンシャル番号）の名前が付けられます。4DRequestsLogファイルサイズが10MBに達するとそのファイルは閉じられ、インクリメントしたシーケンス番号を使用して新しいファイルが作成されます。もし同じ名前のファイルが存在する場合は置き換えられます。シーケンシャル番号の開始番号を、value引数を使用して指定できます。 このテキストファイルにはそれぞれのリクエストに関する様々な情報（時間、プロセス番号、ユーザ、リクエストサイズ、処理時間など）がシンプルな表形式のフォーマットで保存されます。この情報はアプリケーション開発フェーズや統計の目的で利用する場合に特に役立ちます。
Query By Formula On Server	倍長整数	46	**スコープ**: カレントテーブルおよびプロセス **2セッション間で設定を保持**: No **とりうる値**: 0 (データベース設定を使用), 1 (クライアント上で実行) または 2 (サーバ上で実行) **説明**: 引数に渡されたtableに対して実行されるQUERY BY FORMULAやQUERY SELECTION BY FORMULAコマンドの実行場所。 クライアント/サーバモードでデータベースを使用するとき、フォーミュラを使用したクエリをサーバ上またはクライアント上で実行させることができます: • 4D v11 SQLで作成されたデータベースでは、これらのコマンドはサーバ上で実行されます。 • 変換されたデータベースでは、これらのコマンドは、以前のバージョンの4Dと同様、クライアントマシン上で実行されます。 • 変換されたデータベースでは、環境設定のアプリケーション/互換性ページで、これらのコマンドの実行場所をグローバルに変更できます。 この実行場所の違いは、アプリケーションのパフォーマンス (通常サーバ上で実行したほうが早い) だけでなく、プログラミングにも影響します。実際フォーミュラの部品の値 (特にメソッドから呼ばれる変数) は、実行コンテキストにより異なります。このセレクタを使用して開発者は、アプリケーションの動作を適応させられます。 value 引数に0を渡すと、フォーミュラを使用するクエリの実行場所は、データベースの設定に基づきます: 4D v11 SQLで作成されたデータベースでは、これらのコマンドはサーバ上で実行されます。変換されたデータベースでは、データベース環境設定に基づき、クライアントマシンまたはサーバマシンで実行されます。valueに1または2を渡すと、これらのコマンドの実行

			場所をクライアントマシンまたはサーバマシンに強制できます。 例題4を参照してください。 **Note:** "SQLタイプ"のJOINを有効にしたい場合 (QUERY BY FORMULA Joins (49) セレクタ参照) は、フォーミュラがレコードにアクセスすることができるよう、常にフォーミュラをサーバ上で実行しなければなりません。このコンテキストでは、フォーミュラはいかなるメソッド呼び出しも含んではいけないことに注意してください。そうでない場合、自動でリモートマシンでの実行に切り替わります。
Order By Formula On Server	倍長整数	47	**スコープ**: カレントテーブルおよびプロセス **2セッション間で設定を保持**: No **とりうる値**: 0 (データベース設定を使用), 1 (クライアント上で実行) または 2 (サーバ上で実行) **説明**: 引数に渡されたtableに対して実行されるORDER BY FORMULAコマンドの実行場所。 クライアント/サーバモードでデータベースを使用するとき、ORDER BY FORMULAコマンドをサーバ上またはクライアント上で実行させることができます。このセレクタを使用して、このコマンドの実行場所 (サーバまたはクライアント) を指定できます。このモードはデータベース環境設定でも設定できます。詳細はセレクタ46、Query By Formula On Serverの説明を参照してください。 **Note:** "SQLタイプ"のJOINを有効にしたい場合 (QUERY BY FORMULA Joins (49) セレクタ参照) は、フォーミュラがレコードにアクセスすることができるよう、常にフォーミュラをサーバ上で実行しなければなりません。このコンテキストでは、フォーミュ ラはいかなるメソッド呼び出しも含んではいけないことに注意してください。そうでない場合、自動でリモートマシンでの実行に切り替わります。
Auto Synchro Resources Folder	倍長整数	48	**スコープ**: 4D リモートマシン **2セッション間で設定を保持**: No **とりうる値**: 0 (同期しない), 1 (自動同期) または2 (確認する). **説明**: このコマンドを実行する4DクライアントマシンのResourcesフォルダの動的な同期モード。 サーバ上のResourcesフォルダの内容が更新されたり、(リソースエクスプローラや SET DATABASE LOCALIZATION コマンドで) ユーザが同期をリクエストすると、サーバは接続されたユーザに通知を行います。 クライアント側では3つの同期モードを選択できます。Auto Synchro Resources Folderセレクタはカレントセッションでクライアントマシンが使用するモードを指定できます: • 0 (デフォルト値): 動的な同期を行わない (同期リクエストは無視される) • 1: 自動の動的同期 • 2: クライアントマシンにダイアログを表示し、同期の受け入れ回避を確認する アプリケーションの環境設定で、同期モードをグローバルに設定できます。
QUERY BY FORMULA Joins	倍長整数	49	**スコープ**: カレントプロセス **2セッション間で設定を保持**: No **とりうる値**: 0 (データベース設定を使用), 1 (常に自動リレーションを使用) または 2 (可能ならSQL JOINを使用) **説明**: "SQL JOIN"の利用に関連する、QUERY BY FORMULAとQUERY SELECTION BY FORMULAコマンドの動作モード。 4D v11 SQLのバージョン11.2以降で作成されたデータベースでは、これらのコマンドはSQL JOINモデルに基づくJOINを実行します。このメカニズムを使用して、(以前のバージョンでは必要な条件だった) 自動リレーションで接続されていない他のテーブルに対して実行されたクエリに基づき、テーブルのセレクションを変更できます。 QUERY BY FORMULA Joinsセレクタで、カレントプロセスの、フォーミュラによるクエリの動作モードを指定できます: • 0: データベースの現在の設定を使用する (デフォルト値)。4D v11 SQLのバージョン11.2以降で作成されたデータベースでは、"SQL JOIN"はフォーミュラによるクエリに対し常に有効です。変換されたデータベースでは、互換性のためデフォルトでこのメカニズムは有効になっていません。しかし環境設定で実装できます。 • 1: 常に自動リレーションを使用 (= 以前のバージョンの4Dの動作)。このモードでは、他のテーブルに対して行われたクエリに基づきテーブルのセレクションを作成するためには、リレーションが必要です。4Dは"SQL JOIN"を行いません。 • 2: 可能であればSQL JOINを使用 (= 4D v11 SQLのバージョン11.2以降で作成されたデータベースのデフォルト動作)。このモードでは、フォーミュラが適合する限り、4Dはフォーミュラによるクエリに"SQL JOIN"を使用します (2つの例外があります。QUERY BY FORMULAやQUERY SELECTION BY FORMULA コマンドの説明を参照)。 **Note:** 4Dのリモートモードにおいて、"SQL JOIN"はフォーミュラがサーバ上で実行されているときにのみ使用できます (レコードへのアクセスが必要なため)。フォーミュラの実行場所を設定するには、セレクタ46と47を参照してください。
HTTP Compression Level	倍長整数	50	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
HTTP Compression Threshold	倍長整数	51	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Server Base Process Stack Size	倍長整数	53	**スコープ**: 4D Server **2セッション間で設定を保持**: No **とりうる値**:正の倍長整数 **説明**: サーバ上のプリエンプティブシステムプロセス毎に割り当てるスタックのサイズ (バイト単位) です。デフォルトでこの値は1,000,000 (1 MB) です。 プリエンプティブシステムプロセスはメインの4D クライアントプロセスを制御するためにロードされます。デフォルトでそれぞれのプリエンプティブプロセスに割り当てられるサイズはおよその場合最適なサイズですが、何百ものプロセスが作成されるようなケースではこのサイズが適切かどうか検討する必要が出てくるかもしれません。 データベースが実行する処理がそれを許す限り、最適化の目的でこのサイズを大幅に減らすことができます (例えばデータベースで大量のレコードの並び替えなどを行わない場合)。512 や256 KB でさえも設定可能です。 スタックサイズを小さくしすぎることは致命的であり、4D Server の動作に害を及ぼすことになるので注意してください。このパラメタの設定は注意を持って行い、データベースの利用状況 (レコード数や行う処理など) を考慮しながら行わなければなりません。 このパラメタの設定を行うには、On Server Startup データベースメソッドなどを使用してサーバ上でコマンドが実行されなければなりません。
Idle Connections Timeout	倍長整数	54	**スコープ**: 値が負数なら4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**: 持続時間を秒で表す値。値は正数 (新規接続) または負数 (既存の接続)をとることができます。デフォルト値は4D v11 SQLの場合0 (タイムアウトなし) で、4D v12の場合20です。 **説明**: この引数を使用して、4DデータベースエンジンとSQLエンジン両方への動きのない接続の最大時間 (タイムアウト) を設定できます。この設定は、コマンドが実行されたマシンにより開かれたすべての接続に適用されます。 動作していない接続がこの制限時間に達すると、接続は自動でスタンバイ状態に置かれます。つまりクライアント/サーバセッションがフリーズされ、ネットワークソケットが閉じられます。この動作はユーザに対し完全に透過的です。スタンバイ状態の接続でリクエストが開始されると、ソケットが自動で再び開かれ、クライアント/サーバセッションが再び有効になります。 この設定によりサーバのリソースを節約できます。スタンバイ状態の接続はソケットを閉じ、サーバ上のプロセスを解放します。他方これにより、ファイアウォールがアイドルなソケットを閉じてしまうことに伴い接続が失われることを避けることができます。このためには、アイドル接続のタイムアウト値はファイアウォールのタイムアウト値よりも小さくなくてはなりません。 valueに正数を渡すと、設定はすべてのプロセスのすべての新規接続に適用されます。負数を渡すと、設定はカレントプロセスの開かれた接続に適用されます。0を渡すと、アイドル接続のタイムアウトは行われません。 4D v11 SQLの場合、このパラメターはサーバ上でのみ考慮されます。 4D v12の場合、このパラメターはサーバーおよびクライアント両側で設定できます。2つの異なる間隔を設定すると、短いほうが使用されます。通常この値を変更する必要はありません。
PHP Interpreter IP address	倍長整数	55	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: "nnn.nnn.nnn.nnn" (例 "127.0.0.1") のようなフォーマット文字列 **説明**: FastCGI を経由してPHPインタプリタと通信を行うために、4D がローカルで使用するIP アドレス。デフォルトで値は"127.0.0.1" です。このアドレスは4D が配置されているマシンに対応しなければなりません。このパラメタはデータベース設定を使用してすべてのマシン用にグローバルに設定できます PHPインタプリタに関する詳細は*Design Reference*マニュアルを参照してください。
PHP Interpreter port	倍長整数	56	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: 正の倍長整数値。デフォルト値は8002。 **説明**: 4DのPHPインタプリタが使用するTCPポート番号。データベース設定でこの値をすべてのマシン用にグローバルに変更できます。PHPインタープリタに関する詳細は*Design Reference* マニュアルを参照してください。
PHP Number of children	倍長整数	57	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: 正の倍長整数値。デフォルト値は5。 **説明**: 4DのPHPインタプリタがローカルで作成し、管理する子プロセスの数。最適化の目的で、スクリプト実行リクエストを処理するために、PHPインタプリタは" 子プロセス" と呼ばれるシステムプロセスのセット (プール) を作成、使用します。アプリケーションのニーズに基づき、子プロセス数の数を変更できます。データベース設定を使用してこの値をすべてのマシン用にグローバルに設定できます。PHPインタプリタに関する詳細は*Design Reference* マニュアルを参照してください。 **Note:** Mac OS では、すべての子プロセスは同じポートを共有します。Windows では、それぞれの子プロセスが個別のポート番号を使用します。最初の番号はPHP インタプリタ用に設定された番号で、他の子プロセスは最初の番号をインクリメントします。例えばデフォルトポート番号が8002で、5個の子プロセスを起動すると、ポート8002から8006が使用されます。
PHP Max requests	倍長整数	58	**スコープ**: 4D application **2セッション間で設定を保持**: No **値**: 正の倍長整数地。デフォルト値は500。 **説明**: PHP インタプリタが受け入れるリクエストの最大数。この最大値に達すると、インタプリタは"server busy"タイプのエラーを返します。セキュリティおよびパフォーマンスのため、この値を変更できます。データベース設定を使用してすべてのマシン用にグローバルに設定を変更できます。このパラメタに関する詳細はFastCGIPHPのドキュメントを参照してください。 **Note:** 4D側では、これらの引数は動的に適用されます。設定を有効にするために4Dを終了する必要はありません。他方、PHPインタプリタが既に起動されている場合、これらの設定を有効にするためにはインタプリタを再起動しなければなりません。
PHP Use external interpreter	倍長整数	60	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: 0 = 内部インタプリタを使用, 1 = 外部インタプリタを使用 説明: 4D のPHPリクエストを4Dの内部インタプリタに送信するか、外部インタプリタに送信するかを指定する値。デフォルトで値は0 (4Dのインタプリタを使用) です。独自にインストールしたPHPインタプリタを使用したい場合、例えば追加のモジュールを使用したり特定の設定を使用したい場合は、valueに1を渡します。この場合、4DはPHPリクエストの場合でも内部インタプリタを起動しません。 カスタムPHPインタプリタはFastCGIでコンパイルされ、4Dエンジンと同じマシン上に配置されていなければなりません。この場合、開発者がインタプリタを完全に管理しなければならない点に留意してください。4Dはインタプリタを開始したり停止したりしません。 データベース設定を使用してこの値をすべてのマシン用にグローバルに設定できます。
Maximum Temporary Memory Size	倍長整数	61	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**:正の倍長整数値 **説明**: 4D がそれぞれのプロセスに割り当てることのできる一時的なメモリの最大サイズ (MB)。デフォルトで値は 0 (最大サイズの設定なし) です。 4D はインデックスやソート処理のために特別な一時的メモリを使用します。このメモリは大量の処理を行う間、" 標準" キャッシュメモリの保護を意図したものです。これは必要な時にのみ有効になります。デフォルトで、一時的なメモリのサイズは、(システムメモリ設定に基づく) 利用可能なリソースによってのみ制限されます。 このメカニズムはほとんどのアプリケーションで適しています。しかし特定の特別なコンテキスト、特に同時に多数のシーケンシャルソートを行うようなサーバ/ クライアントアプリケーションでは、一時的なメモリのサイズが、システムが不安定になるほどに致命的に増加するかもしれません。このような場合は、一時的メモリの最大サイズを設定することで、アプリケーションが正しく動作するようにできます。その代わり、実行速度に影響が出ます。プロセスに対する最大サイズに達すると、4D はディスクファイルを使用し、そのために処理が遅くなります。 先のようなケースの場合、だいたい50 MB が一般的なサイズとしてよいと思われます。しかし適切な値はアプリケーションの特性、そして実際の環境でのテスト結果に基づき決定されるべきです。
SSL Cipher List	文字列	64	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**: コロンで区切られた一連の文字列 (例 "RC4-MD5:RC4-64-MD5:....") **説明**: 暗号リストはSSLプロトコルのために4Dが使用します。例えばこのセレクターを使用してSSL 3.0暗号化アルゴリズムを実装でき、そしてSSL 2.0による接続を拒否できます。 この設定はアプリケーション全体に適用されます (HTTP サーバーやSQLサーバー、およびSSL プロトコルを使用するすべての4D の機能に関連) が、その効果は一時的であり、セッションをまたいでは保持されません。 暗号リストが変更されると、新しい設定を有効にするために、関連するサーバーを再起動しなければなりません。 暗号リストを (SLIファイルに恒久的に格納された) デフォルト値に再設定するには、value引数に空の文字列 ("") を渡してSET DATABASE PARAMETERコマンドを呼び出します。 デフォルトで、4DはRC4暗号化アルゴリズムを使用します。(より最新の) AESアルゴリズムを使用するには、value引数に以下の文字列を渡します: "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH" **Note**: Get database parameterコマンドで暗号リストはオプションのstringValue引数に返され、戻り値は常に0となります。
Cache unload minimum size	倍長整数	66	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**: 1より大きい正の倍長整数 **説明**: エンジンがオブジェクトをデータベースキャッシュに配置する際に空き空間を作成する必要が出た場合、データベースキャッシュからリリースするメモリの最小サイズ (バイト単位)。 このセレクタの目的はキャッシュからデータをリリースする時間を減らし、よりよりパフォーマンスを得ることにあります。キャッシュのサイズやデータベース中で処理されるデータのブロックサイズに応じてこの値を変更できます。 このセレクタが使用されないとデフォルトで、4Dは空間が必要になった時最低10%のキャッシュをアンロードします。
Direct2D Status	倍長整数	69	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **説明**: WindowsにおけるDirect2D実装のアクティベーションモード **取りうる値**: 以下の定数のいずれか (デフォルトでモード5): Direct2D Disabled (0): Direct2Dモードは無効であり、データベースは過去のモード (GDI/GDIPlus) で動作する。 Direct2D Hardware (1): 4Dアプリケーション全体でDirect2Dグラフィックハードウェアコンテキストを使用する。このコンテキストが利用できない場合、Direct2Dグラフィックソフトウェアコンテキストを使用 (Vistaを除く。Vistaではパフォーマンスのために GDI/GDIPlusモードが使用されます)。 Direct2D Hardware SVG and Editors (2): SVG、コード、およびストラクチャーエディターでDirect2Dグラフィックハードウェアコンテキストを使用する。このコンテキストが利用できない場合、Direct2Dグラフィックソフトウェアコンテキストを使用 (Vistaを除く。VistaではパフォーマンスのためにGDI/GDIPlusモードが使用されます)。 Direct2D Software (3): Windows 7以降、4Dアプリケーション全体でDirect2Dグラフィックソフトウェアコンテキストを利用。VistaではパフォーマンスのためにGDI/GDIPlusモードが使用されます。 Direct2D Software SVG and Editors (4): Windows 7以降、SVG、コード、およびストラクチャーエディターのDirect2Dグラフィックソフトウェアコンテキストを使用する。VistaではパフォーマンスのためにGDI/GDI/GDIPlusモードが使用されます。 Direct2D Mixed (5) (デフォルトモード): Windows 7以降、SVG、コード、およびストラクチャーエディターでDirect2Dグラフィックハードウェアコンテキストを使用する。4Dアプリケーションの他の部分ではDirect2Dグラフィックソフトウェアコンテキストを使用する。VistaではパフォーマンスのためGDI/GDIPlusモードを使用します。

Direct2D Get active status	倍長整数	74	**注:** このセレクターはGet database parameterコマンドでのみ使用することができ、設定することはできません。
			説明: WindowsにおいてDirect2Dのアクティブな実装を返します。
			取りうる値: 0, 1, 2, 3, 4 または 5 (セレクター69の値参照)。返される値はDirect2Dが利用可能かどうか、およびハードウェア、OSによってサポートされるDirec2Dの品質に基づきます。
			例えば以下のコードを実行した場合、
			`SET DATABASE PARAMETER(Direct2D Status;Direct2D Hardware)`
			`$mode:=Get database parameter(Direct2D Get active status)`
			- Windows 7以降、システムがDirect2D互換のハードウェアを検知すると、$modeに1が設定されます。そうでなければ$modeは3に設定されます (ソフトウェアコンテキスト)。
			- Windows Vistaでは、システムがDirect2D互換のハードウェアを検知すると、$modeに1が設定されます。そうでなければ$modeは0に設定されます (Direct2D無効)。
			- Windows XPでは、$modeは常に0です (Direct2D非互換)。

Diagnostic Log Recording	倍長整数	79	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **取りうる値**: 0 または 1 (0 = 記録しない, 1 = 記録する) **説明**: 4D診断ファイルの記録を開始または停止する。デフォルトの値は0で記録を行いません。 4Dは診断ファイルの中に内部的なアプリケーション処理に関連するイベントを継続的に記録することができます。このファイルに含まれる情報は4Dアプリケーション開発のために使用され、4D社の技術サポート担当により解析されます。このセレクターに1を渡すと、DatabaseName.txtという名称のファイルが自動でデータベースの**Logs**フォルダーに作成されるか、既に存在する場合は開かれます。このファイルのサイズが10MBに達するとそのファイルは閉じられ、DatabaseNameN.txtが生成されます (Nは連番)。 LOG EVENTコマンドを使用してカスタム情報をこのファイルに書き込むこともできます。

Log Command list	文字列	80	**スコープ**: 4D アプリケーション
			2セッション間で設定を保持: No
			取りうる値: 記録する4Dコマンドの番号リスト。型は文字列で各コマンド番号をセミコロンで区切ります。"all"を渡すとすべてのコマンドが記録され、"" (空の文字列) を渡すとなにも記録されません。
			説明: デバッグファイルに記録する4Dコマンドのリスト (セレクター 34, Debug Log Recording参照)。デフォルトではすべての4Dコマンドが記録されます。
			このセレクターを使用すれば、記録に残したい4Dコマンドを指定することで、デバッグファイルに保存される情報の量を制限することができます。例えば以下のようにコードを記述できます:
			`SET DATABASE PARAMETER(Log Command list;"277;341")`
			`// QUERY および QUERY SELECTION コマンドのみを記録する`

Spellchecker	倍長整数	81	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **とりうる値**: 0 (default) = OS Xのシステムのスペルチェッカー(ハンスペルは無効化されます), 1 = ハンスペルスペルチェッカー有効化 **説明**: OS X環境下においてハンスペルスペルチェッカーを有効化します。デフォルトでは、このプラットフォームではシステムのスペルチェッカーが有効化されています。例えば、クロスプラットフォームアプリケーションのインターフェースを統一するためにハンスペルを使用したいという場面があるかもしれません(Windowsでは、ハンスペルチェッカーのみが動作します)。詳細な情報に関しては、Hunspell辞書のサポートを参照して下さい。
QuickTime support	倍長整数	82	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: Yes **値**: 0 (default) = QuickTime無効, 1 = QuickTime有効 **説明**: v14以降、4DではQuickTimeコーデックはサポートされなくなりました。互換性のために、このセレクターを使用してデータベース内でQuickTimeを再有効化することができます。このオプションの変更にはデータベースの再起動が必要です。なお、将来の4DのバージョンではQuickTimeサポートは恒久的に廃止されることに注意して下さい。
JSON use local time	倍長整数	85	

注: table引数はセレクター31, 46, そして47で使用されます。それ以外の場合、この引数は無視されます。

例題 1

以下の文は予期しないタイムアウトを避けます:

```
 `カレントプロセスのタイムアウトを3時間にします
SET DATABASE PARAMETER(4D Server Timeout;-60*3)
 `4Dからコントロールできない時間のかかる処理を行う
...
WR PRINT MERGE(Area;3;0)
...
```

例題 2

この例題は、一時的にクライアントマシン上でフォーミュラによるクエリを実行するよう強制します:

```
curVal:=Get database parameter([table1];Query By Formula On Server) `現在の設置を取得
SET DATABASE PARAMETER([table1];Query By Formula On Server;2) `クライアントマシンでの実行を強制
QUERY BY FORMULA([table1];myformula)
SET DATABASE PARAMETER([table1];Query By Formula On Server;curVal) `設定を元に戻す
```

例題 3

変換された4D date を含む JSON形式のデータを書き出したい場合を考えます。変換が起きるのは日付がオブジェクトに保存されたときなので、OB SETコマンドが呼び出される前に SET DATABASE PARAMETERコマンドを使用する必要があることに注意して下さい:

```
C_OBJECT($o)
SET DATABASE PARAMETER(JSON use local time;0)
OB SET($o ;"myDate";Current date) // JSON への変換
$json:=JSON Stringify($o)
SET DATABASE PARAMETER(JSON use local time;1)
```

SET UPDATE FOLDER

SET UPDATE FOLDER (folderPath {; silentErrors})

引数	型	説明
folderPath	文字	⇒アップデートされたアプリケーションを内包するフォルダ(OS Xの場合はパッケージ)へのパス名
silentErrors	ブール	⇒False (デフォルト)=エラーを表示して報告 True=エラーを表示しない

説明

SET UPDATE FOLDERコマンドは、カレントの組み込まれた4Dアプリケーションのアップデート情報を含んだフォルダの場所を指定します。この情報は RESTART 4Dメソッドが呼ばれるまで4Dセッション中は保存されます。アプリケーションが手動で終了した場合、この情報は保存されません。

このコマンドは組み込みアプリ(サーバーアプリまたはシングルユーザー向けアプリ)の自動アップデート処理において使用されるためのコマンドです。詳細な情報に関しては、デザインリファレンスマニュアルの の章を参照して下さい。

Note: このコマンドは4D サーバーか、または4D Volume Desktopのシングルユーザー向け組み込みアプリでしか使用できません。

folderPath 引数には、組み込みアプリの新バージョンのフォルダへの完全なパス名を渡します(Windowsであれば *my4DApp.exe* アプリケーションを内包しているフォルダ、OS Xであれば *my4DApp.app* パッケージを内包しているフォルダ)。新しいバージョンは4D v14のアプリケーションビルダーを使用して生成されている必要があります。特に、4Dに入っていてリモートアップデートを管理するために使用されている"アップデーター"ツールのカレントバージョンを内包している必要があります。

Note: アプリケーションフォルダはアップデートの際に上書きされるため、アプリケーションの新バージョンのファイル名は、オリジナルのものと同じ名前を使用することが推奨されます。異なる名前を使用した場合、保存されたショートカットやパスは全て動かなくなります。

引数が有効であれば、セッション中は RESTART 4D コマンドが呼び出されるまでアップデートは"on hold"となります。RESTART 4D を呼び出す前に **SET UPDATE FOLDER**を複数回呼び出した場合、最後の有効な呼び出しのみが有効とみなされます。

例外が起きた場合にはエラーが生成されます。silentErrors 引数によってこのエラーを表示するかどうかを指定することが出来ます(以下を参照)。

folderPath に空の文字列 ("")を渡す事によってカレントセッションにおけるアップデート情報をリセットすることができます。

任意の silentErrors 引数はアップデーターツールからのエラーがどのように報告されるかを指定します。

- **False** を渡すかこの引数を省略した場合、エラーはアップデートジャーナルに記録され、警告ダイアログボックス内にその旨が表示されます。
- **True** を渡した場合、エラーはアップデートジャーナルに記録され、表示はされません。

例外: アップデータージャーナルを作成できないときには、silentErrors 引数の値に関わらず警告ダイアログボックスが表示されます。詳細な情報に関しては、Get last update log path コマンドの詳細を参照して下さい。

コマンドが正常に実行された場合には、OKシステム変数は1に設定されます。それ以外の場合は0に設定されます。このコマンドによって生成されたエラーはどれも ON ERR CALLコマンドを使用して実装されたメソッドによって割り込み可能です。

例題

ディスク上に "MyUpdates"を作成し、その中に "MyApp"アプリケーションの新しいバージョンが入っているとします。また、エラーは表示したくないとします。このアップデートのためには、以下の様に記載します。

```
  // Windows syntax
 SET UPDATE FOLDER("C:\\MyUpdates"+Folder separator+"MyApp";True)

  // OS X syntax
 SET UPDATE FOLDER("MacHD:MyUpdates"+Folder separator+"MyApp.app";True)
```

Structure file

Structure file {(*)} -> 戻り値

引数	型		説明
*	演算子	→	ホストデータベースのストラクチャファイルを返す
戻り値	文字	↩	データベースストラクチャファイルのパス名

説明

Structure file コマンドは、現在使用しているデータベースのストラクチャファイルのパス名を返します。

Windows
例えば、ボリュームG上の\DOCS\MyCDsの中に配置されたMyCDsデータベースを使って作業している場合、この関数は、
G:\DOCS\MyCDs.4dbase\MyCDs.4DBを返します。

Macintosh
例えば、ハードディスクMacintosh HD上のDocuments:MyCDs:フォルダの中に配置されたMyCDsデータベースを使って作業している場合、この関数は、Macintosh HD:Documents:MyCDs.4dbase:MyCDs.4DBを返します。

Note: データベースがコンパイルされて4D Volume Desktopに統合されている場合、WindowsおよびMac OSでこのコマンドはアプリケーションファイル（実行可能なアプリケーション）のパス名を返します。Mac OS上では、このファイルはソフトウェアパッケージの中の[Contents:Mac OS]フォルダに置かれます。これは以前のメカニズムに起因するものであり、互換性のため保持されています。ソフトウェアパッケージ自体のフルアクセスパスを取得したい場合には、Application fileコマンドの利用をお勧めします。方法としては、Application type コマンドを使用してアプリケーションタイプを調べた後、その結果に応じてStructure fileまたはApplication fileを実行します。

警告： 4D Clientを実行している最中にこの関数を呼び出すと、パス名ではなくストラクチャファイルの名前だけが返されます。

オプションの * 引数はコンポーネントを使用したアーキテクチャで有用です。コマンドが呼び出されたコンテキストで、ホストデータベースまたはコンポーネントどちらのストラクチャファイルのパス名を取得するか指定するために、使用できます:

コマンドがコンポーネントから呼び出された場合:

- * 引数が渡されていると、コマンドはホストデータベースのストラクチャファイルのパス名を返します。

- * 引数が渡されないと、コマンドはコンポーネントのストラクチャファイルのパス名を返します。

コンポーネントのストラクチャファイルは、データベースの"Components"フォルダに置かれた.4dbまたは.4dcファイルに対応します。しかしコンポーネントはエイリアス/ショートカット、または.4dbaseフォルダ/パッケージでインストールすることもできます:

- エイリアス/ショートカットでインストールされたコンポーネントの場合、コマンドはオリジナルの.4dbまたは.4dcファイルのパスを返します (エイリアスやショートカットは解決されます)。

- .4dbaseフォルダ/パッケージでインストールされたコンポーネントの場合、コマンドはこのフォルダ/パッケージ内の.4dbまたは.4dcファイルのパスを返します。

ホストデータベースのメソッドからコマンドが呼ばれた場合、常にホストデータベースのストラクチャのパス名を返します。* 引数は無視されます。

例題 1

以下の例は、現在使用中のストラクチャファイルの名前と配置場所を表示します:

```
 If(Application type#4D Remote Mode)
    $vsStructureFilename:=Long name to file name(Structure file)
    $vsStructurePathname:=Long name to path name(Structure file)
    ALERT("You are currently using the database "+Char(34)+$vsStructureFilename+Char(34)+" located at
"+Char(34)+$vsStructurePathname+Char(34)+".")
 Else
    ALERT("You are connected to the database "+Char(34)+Structure file+Char(34))
 End if
```

Note: プロジェクトメソッドLong name to file nameとLong name to path nameはシステムドキュメント の節で紹介しています。

例題 2

以下の例題は、メソッドがコンポーネントから呼び出されているか知るために使用できます:

```
 C_BOOLEAN($0)
 $0:=(Structure file#Structure file(*))
  ` $0=メソッドがコンポーネントから呼び出されている場合、True
```

VERIFY CURRENT DATA FILE

VERIFY CURRENT DATA FILE {(objects ; options ; method {; tablesArray {; fieldsArray}})}

引数	型		説明
objects	倍長整数	→	検証するオブジェクト
options	倍長整数	→	検証オプション
method	テキスト	→	4Dコールバックメソッド名
tablesArray	倍長整数配列	→	検証するテーブル番号
fieldsArray	2D整数配列, 2D倍長整数配列, 2D実数配列	→	検証するインデックス番号

説明

VERIFY CURRENT DATA FILEコマンドは、4Dが現在開いているデータファイル中にあるオブジェクトの、構造的な検証を行います。

このコマンドは、開かれたデータベースのカレントのデータファイルのみに適用されることを除き、VERIFY DATA FILEコマンドと同じ機能を持ちます。そのため、ストラクチャとデータを指定する引数は必要ありません。
引数の説明はVERIFY DATA FILEコマンドを参照してください。

VERIFY CURRENT DATA FILEコマンドを引数なしで呼び出すと、デフォルトの設定値を使用して検証が行われます:

- objects = Verify All (= 値 16)
- options = 0 (ログファイルを作成する)
- method = ""
- tablesArrayとfieldsArrayは省略される.

このコマンドが実行されると、データキャッシュがフラッシュされ、検証中はデータにアクセスするすべての操作がブロックされます。

VERIFY DATA FILE

VERIFY DATA FILE (structurePath ; dataPath ; objects ; options ; method {; tablesArray {; fieldsArray}})

引数	型		説明
structurePath	テキスト	→	検証する4Dストラクチャファイルのパス名
dataPath	テキスト	→	検証する4Dデータファイルのパス名
objects	倍長整数	→	検証するオブジェクト
options	倍長整数	→	検証オプション
method	テキスト	→	4Dコールバックメソッド名
tablesArray	倍長整数配列	→	検証するテーブル番号
fieldsArray	2D整数配列, 2D倍長整数配列, 2D実数配列	→	検証するインデックス番号

説明

VERIFY DATA FILEコマンドは、structurePathとdataPathで指定したデータファイル中にあるオブジェクトの、構造的な検証を行います。

注: データ検証に関する詳細は、Design Referenceマニュアルを参照してください。

structurePathは、検証するデータファイルに対応するストラクチャファイル (コンパイル済みまたはインタプリタ) を指定します。開かれたストラクチャや他のストラクチャを指定できます。OSに対応した完全パス名を指定しなければなりません。空の文字列を渡すと標準のファイルを開くダイアログボックスが表示され、使用するストラクチャファイルをユーザが指定できます。

dataPath は4Dデータファイル (.4DD) を指定します。データファイルはstructurePath引数で指定されたストラクチャファイルに対応していなければなりません。カレントストラクチャーファイルを指定することができますが、カレントのデータファイルは指定できない (開かれていてはいけない) ことに注意してください。現在開かれているデータファイルを検証するためにはVERIFY CURRENT DATA FILEコマンドを使用します。**VERIFY DATA FILE**でカレントのデータファイルを検証しようとすると、エラーが生成されます。

指定されたデータファイルは読み込みのみで開かれます。他のアプリケーションが書き込み可能でこのファイルにアクセスしないようにしなければなりません。そうでなければ検証結果は正しくないものになります。
dataPath引数に空の文字列、ファイル名、またはOSのシンタックスに対応した完全パス名を渡すことができます。空の文字列を渡すと標準のファイルを開くダイアログボックスが表示され、検証するファイルをユーザが選択できます。カレントのデータファイルを選択できないことに注意してください。データファイル名のみを渡した場合、4Dは指定されたストラクチャファイルと同階層にあるデータファイルを探します。

objects 引数は検証するオブジェクトを指定するために使用します。2つのタイプのオブジェクト、レコードとインデックスを検証できます。Data File Maintenanceテーマの以下の定数を使用できます:

定数	型	値	コメント
Verify All	倍長整数	16	
Verify Indexes	倍長整数	8	このオプションを使用すると、インデックスの物理的な整合性を検証しますが、データとのリンクは考慮されません。この検証は無効なキーを検知しますが、重複キー (同じレコードを参照する2つのインデックス) を検出することはできません。この検証を行うにはVerify Allオプションを使用しなければなりません。
Verify Records	倍長整数	4	

レコードとインデックス両方を検証するにはVerify Records+Verify Indexesを渡します。0を渡しても同じ結果が得られます。Verify Allオプションを指定すると内部的な検証が完全に行われます。この検証はログの作成と互換性があります。

options 引数は検証オプションを設定するために使用します。現時点ではData File Maintenanceテーマの1つのオプションのみを指定できます:

定数	型	値	コメント
Do not create log file	倍長整数	16384	通常このコマンドはXMLフォーマットのログファイルを作成します。このオプションを使用すればログファイルは作成されません。

通常、**VERIFY DATA FILE**コマンドはXMLフォーマットのログファイルを作成します (このコマンドの最後の説明を参照してください)。このオプションを指定して、ログの作成をキャンセルできます。ログファイルを作成するには、optionsに0を渡します。

method 引数には、検証中定期的に呼び出されるコールバックメソッドを設定するために使用します。空の文字列を渡すと、メソッドは呼び出されません。渡されたメソッドが存在しない場合、検証は行われず、エラーが生成され、OKシステム変数に0が設定されます。コールバックメソッドが呼び出されるときは、検証されるオブジェクトのタイプおよび呼び出し元のイベントタイプにより最大5つの引数が渡されます。コールバックメソッドではこれらの引数を宣言しなければなりません:

$1	倍長整数	メッセージタイプ (表参照)
$2	倍長整数	オブジェクトタイプ
$3	テキスト	メッセージ
$4	倍長整数	テーブル番号
$5	倍長整数	予約

以下の表は、イベントタイプごとの引数の内容を示しています:

イベント	$1 (Longint)	$2 (Longint)	$3 (Text)	$4 (Longint)	$5 (Longint)
メッセージ	1	0	進行状況メッセージ	処理率 (0-100%)	予約
検証終了 (*)	2	オブジェクトタイプ (**)	OKメッセージテスト	テーブルまたはインデックス番号	予約
エラー	3	オブジェクトタイプ (**)	エラーテキスト-メッセージ	テーブルまたはインデックス番号	予約
実行終了	4	0	終了	0	予約
警告	5	オブジェクトタイプ (**)	エラーテキストメッセージ	テーブルまたはインデックス番号	予約

(*) モードがVerify Allであるとき、検証終了 ($1=2) が返されることはありません。このメッセージはVerify Records または Verify Indexes モードでのみ使用されます。
(**) *Object type*: オブジェクトタイプ: オブジェクトが検証されると、OKメッセージ ($1=2)、エラー ($1=3)、警告 ($1=5) が送信されます。$2に返されるオブジェクトタイプは以下のうちのいずれかになります:

- 0 = 不明
- 4 = レコード
- 8 = インデックス
- 16 = ストラクチャオブジェクト (データファイルの予備検証)

特別なケース: $1が2、3、または5のとき、$4が0ならば、それはメッセージがテーブルやインデックスについてではなく、データファイル全体に関するものであることを示します。

コールバックメソッドは$0に倍長整数値を返さなくてはなりません。これは処理の実行をチェックするために使用されます:

- $0 = 0の時、処理は通常通り続行されます。
- $0 = -128の時、処理は停止されますが、エラーは生成されません。
- $0 = 他の値の時、処理が停止され、$0に返された値をエラー番号としてエラーを生成します。このエラーはエラーハンドラメソッドでとらえることができます。

注: 実行終了イベント ($4=1) が生成された後、$0を使用して実行を中断させることはできません。

2つのオプションの配列をこのコマンドで利用できます:

- tablesArray 配列にはテーブル番号が含まれ、レコードを検証するテーブルを指定するために使用します。この引数は検証するテーブルを制限するために使用します。この引数を渡さないか配列が空の場合で、objects引数にVerify Recordsが指定されている場合、すべてのテーブルが検証されます。
- fieldsArray 配列には検証対象とするインデックス付きフィールドの番号を渡します。

この引数が渡されないか配列が空の場合で、objects引数にVerify Indexesが指定されている場合、すべてのインデックスが検証されます。コマンドはインデックスの無いフィールドを無視します。フィールドに複数のインデックスが含まれる場合、すべてが検証されます。フィールドが複合インデックスの一部である場合、インデックス全体が検証されます。
fieldsArrayには二次元配列を渡します。配列の内容は以下の通りです:
- 要素 {0} にはテーブル番号が含まれます。
- 他の要素 {1...x} にはフィールド番号が含まれます。

デフォルトで**VERIFY DATA FILE**コマンドは、(options引数にDo not create log fileオプションが指定されていなければ) XMLフォーマットのログファイルを作成します。ログファイルの名前はデータファイル名に基づきつけられ、データベースの"Logs"に作成されます。たとえば、"data.4dd"データファイルを検証すると、"data_verify_log.xml"が作成されます。

例題 1

データとインデックスの検証:

```
VERIFY DATA FILE($StructName;$DataName;Verify Indexes+Verify Records;Do not create log file;"")
```

例題 2

完全な検証を行い、ログを作成する:

```
VERIFY DATA FILE($StructName;$DataName;Verify All No Callback;0;"")
```

例題 3

レコードのみの検証:

```
VERIFY DATA FILE($StructName;$DataName;Verify Records;0;"")
```

例題 4

テーブル3と7のレコードのみを検証:

```
ARRAY LONGINT($arrTableNums;2)
$arrTableNums{1}:=3
$arrTableNums{2}:=7
VERIFY DATA FILE($StructName;$DataName;Verify Records;0;"FollowScan";$arrTableNums)
```

例題 5

特定のインデックスを検証 ([table4]field1、[table5]field2とfield3):

```
ARRAY LONGINT($arrTableNums;0) `使用しないが必須
ARRAY LONGINT($arrIndex;2;0) `2行 (列は後で追加)
$arrIndex{1}{0}:=4 ` 要素0にテーブル番号
APPEND TO ARRAY($arrIndex{1};1) `検証する1番目のフィールド番号
$arrIndex{2}{0}:=5 ` 要素0にテーブル番号
APPEND TO ARRAY($arrIndex{2};2) ` 検証する1番目のフィールド番号
APPEND TO ARRAY($arrIndex{2};3) ` 検証する2番目のフィールド番号
VERIFY DATA FILE($StructName;$DataName;Verify Indexes;0;"FollowScan";$arrTableNums;$arrIndex)
```

システム変数およびセット

コールバックメソッドが存在しない場合、エラーが生成され、OKシステム変数には0が設定されます。

Version type

Version type -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	デモまたはフルバージョン、64-bitまたは32-bitバージョン

説明

Version type コマンドは、現在実行している4Dや4D Server環境のバージョンタイプを示す数値を返します。4Dでは 4D Environmentテーマ内にある以下の定義済み定数が用意されています。

定数	型	値	コメント
64 bit Version	倍長整数	1	
Demo Version	倍長整数	0	
Merged application	倍長整数	2	バージョンは 4D Volume Desktopを使用して組み込まれたアプリケーションです。

注: カレントバージョンの4Dでは、デモモードは使用することが出来ません。

注: **Version type** は*ビットフィールド*の形式で値を返します。ビットワイズ演算子を使用してそれを解析しなければなりません (例題参照)。

互換性の注: v13.2以前の4Dでは、このコマンドに対して異なるセットの定数が用意されていました。しかしながら、これらの定数は一部の場合において正しく動作しなかったため変更されています。この変更により、既存のコードがアップデートされる必要性があります(例題参照)。しかしながら、以前の挙動のままでいてほしいのであれば、既存のコード内の定数を以前の値(2:64ビットバージョン、1:デモバージョン、0:フルバージョン)で置き換えて下さい。

例題 1

現在の4Dアプリケーションのバージョンタイプを取得するには以下のコードを使用します:

```
If(Version type?? 0)
 // デモバージョン 用のコード
Else
 // フルバージョン 用のコード
End if
If(Version type?? 1)
 // 64-bit 用のコード
Else
 // 32-bit 用のコード
End if
```

例題 2

以下の例によって、あるバージョンが4Dによってビルドされたアプリなのか4Dサーバーによって開かれたデータベースなのかを判別し、それに応じて違うコードを実行する、ということが出来るようになります:

```
If(Version type?? Merged application)
 // ここにビルドされたアプリ用のコードを書きます。
Else
 // ここに4Dによって実行されたデータベース用のコードを書きます。
End if
```

ADD DATA SEGMENT

ADD DATA SEGMENT
このコマンドは引数を必要としません

説明

互換性に関する注記: 4Dバージョン11より、データセグメントはサポートされていません (データファイルのサイズに制限はありません)。このコマンドは呼び出されても何も行いません。

DATA SEGMENT LIST

DATA SEGMENT LIST (Segments)

引数	型		説明
Segments	文字配列	←	データベースのデータセグメントのパス名

説明

DATA SEGMENT LIST コマンドは、segments配列に、現在使用しているデータベースのデータセグメントのパス名を返します。

互換性メモ: 4Dバージョン11より、データセグメントはサポートされなくなりました (データファイルのサイズ制限がなくなりました)。このコマンドは互換性のために残されており、データベースのデータファイルのパス名を一つだけ返します。

BLOB

BLOBコマンド
BLOB PROPERTIES
BLOB size
BLOB TO DOCUMENT
BLOB to integer
BLOB to list
BLOB to longint
BLOB to real
BLOB to text
BLOB TO VARIABLE Updated 14.0
COMPRESS BLOB
COPY BLOB
DECRYPT BLOB
DELETE FROM BLOB
DOCUMENT TO BLOB
ENCRYPT BLOB
EXPAND BLOB
INSERT IN BLOB
INTEGER TO BLOB
LIST TO BLOB
LONGINT TO BLOB
REAL TO BLOB
SET BLOB SIZE
TEXT TO BLOB
VARIABLE TO BLOB Updated 14.0

BLOBコマンド

定義

4Dは、BLOB（Binary Large OBjects）データタイプをサポートします。

BLOBフィールドおよびBLOB変数は、以下のように定義できます。

- BLOBフィールドを作成するには、**インスペクター**ウィンドウ内の**フィールドタイプ**ドロップダウンリストでBLOBを選択します。
- BLOB変数を作成するには、コンパイラー宣言コマンドC_BLOBを使用します。BLOB型のローカル、プロセス、インタープロセス変数を作成できます。

注: BLOBの配列はありません。

4Dの中で、BLOBは連続した可変長バイトであり、1つのまとまったオブジェクトまたは各バイトが個々にアドレス指定できるオブジェクトとして取り扱うことができます。BLOBは空（長さがNULL）でもよく、また最大2,147,483,647バイト（2GB）まで含むことができます。

BLOBとメモリ

BLOBは全体がメモリにロードされます。BLOB変数はメモリ内にだけ保持され、存在します。BLOBフィールドは、そのフィールドが属するレコードの他の部分と同様に、ディスクからメモリにロードされます。

大量のデータを保持できる他のフィールドタイプ（ピクチャー）と同様に、レコードを更新してもBLOBフィールドはメモリに複製されません。その結果、OldおよびModifiedコマンドをBLOBフィールドに適用しても、返される結果は意味を持ちません。

BLOBの表示

BLOBには、どのような種類のデータでも保持できるため、画面上でのデフォルトの表現はありません。フォーム内でBLOBフィールドまたは変数を表示すると、どのような内容であっても常に空白になります。

BLOBフィールド

BLOBフィールドを使用すると、最大で2GBまでのあらゆる種類のデータを保存できます。BLOBフィールドにインデックス付けすることはできないため、BLOBフィールドに保存された値のレコードを検索するには、式を使用しなければなりません。

引数渡し、ポインタ、および戻り値

4DのBLOBは、4Dコマンドまたは4Dプラグインの引数として渡すことができます。BLOBをユーザーメソッドのパラメーターとして渡したり、関数の戻り値にすることもできます。

ポインターを使用して、BLOBをメソッドに渡すことも出来ます。BLOBへのポインターを定義し、ポインターをパラメーターとして渡します。

例題:

```
  ` BLOBタイプの変数を定義
C_BLOB(anyBlobVar)
  ` 4DコマンドにBLOBを引数として渡す
SET BLOB SIZE(anyBlobVar;1024*1024)
  ` プラグインにBLOBを引数で渡す
$errCode:=Do Something With This BLOB(anyBlobVar)
  ` BLOBを引数として渡し、戻り値をBLOBで受け取る。
C_BLOB(retrieveBlob)
retrieveBlob:=Fill_Blob(anyBlobVar)
  ` BLOBのポインターをメソッドに渡す。
COMPUTE BLOB(->anyBlobVar)
```

プラグイン開発者への注意: BLOB引数は"&O"（数字の0ではなく、アルファベットの"O"）として宣言されます。

代入

BLOBを相互に代入できます.

例題:

```
  ` 2つのBLOB変数を定義
C_BLOB(vBlobA;vBlobB)
  ` BLOBに10KBを割り当てる
SET BLOB SIZE(vBlobA;10*1024)
  ` 最初のBLOBを2つめのBLOBに代入
vBlobB:=vBlobA
```

ただし、BLOBに演算子を適用することはできません。BLOBタイプの式はありません。

BLOBのアドレス指定

中カッコ{...}を使用し、BLOBの各バイトを個別にアドレス指定できます。BLOB内では、各バイトに0 から N-1の番号が割り当てられています。NはBLOBのサイズです。例えば:

```
  ` BLOBを定義する
C_BLOB(vBlob)
  ` BLOBのサイズを256バイトに設定する
SET BLOB SIZE(vBlob;256)
  ` 次のループは、256バイトをゼロに初期化する
For(vByte;0;BLOB size(vBlob)-1)
   vBlob{vByte}:=0
End for
```

BLOBの各バイトはすべて個別にアドレス指定できるため、BLOBフィールドまたは変数に格納したいものは実際何でも格納できます。

BLOBと4Dコマンド

4DはBLOBに使用する以下のコマンドを提供します:

- SET BLOB SIZEは、BLOBフィールドや変数のサイズを変更します。
- BLOB sizeは、BLOBのサイズを戻します。
- DOCUMENT TO BLOB や BLOB TO DOCUMENTを使用すると、ドキュメント全体をBLOBからロード、またはBLOBに書き込むことができます（またMacintosh上では、データフォークおよびリソースフォークを選択できます）。
- VARIABLE TO BLOB や BLOB TO VARIABLEコマンドを使用すると、LIST TO BLOB および BLOB to listと同様、4D変数をBLOBに格納、または取り出すことができます。
- COMPRESS BLOB, EXPAND BLOB および BLOB PROPERTIESを使用すると、圧縮されたBLOBを操作できます。
- BLOB to integer, BLOB to longint, BLOB to real, BLOB to text, INTEGER TO BLOB, LONGINT TO BLOB, REAL TO BLOB そして TEXT TO BLOBコマンドを使用すると、ディスク、リソース、OS等から入力される構造化されたデータを操作できます。
- DELETE FROM BLOB, INSERT IN BLOB そして COPY BLOBを使用すると、BLOB内にある大きいサイズのデータのまとまりをすばやく処理することができます。
- ENCRYPT BLOB と DECRYPT BLOBにより4Dデータベース上のデータの暗号化と復号化ができます。

これらのコマンドについては、この章で説明しています。

追記:

- C_BLOBはタイプがBLOBの変数を宣言します。詳細はコンパイラコマンドを参照してください。
- GET PASTEBOARD DATA や APPEND DATA TO PASTEBOARDを使用すると、ペーストボードに格納されているどのデータタイプでも操作できます。詳細はペーストボードの管理を参照してください。
- GET RESOURCE や SET RESOURCEを使用すると、ディスク上に格納されているリソースを操作できます。詳細はCurrent dateを参照してください。
- WEB SEND BLOBはウェブブラウザ上にどのようなタイプのデータでも送ることができます。詳細はWebサーバを参照してください。
- PICTURE TO BLOB, BLOB TO PICTURE そして PICTURE TO GIFにより画像を開いたり、変換することができます。詳細はピクチャを参照してください。
- GENERATE ENCRYPTION KEYPAIR と GENERATE CERTIFICATE REQUESTは、SSL (Secured Socket Layer) セキュア接続プロトコルで使用されるコマンドです。詳細は暗号化プロトコルを参照してください。

BLOB PROPERTIES

BLOB PROPERTIES (BLOB ; compressed {; expandedSize {; currentSize}})

引数	型		説明
BLOB	BLOB	→	情報を取得するBLOB
compressed	倍長整数	←	0 = BLOBは圧縮されていない 1 = BLOBは圧縮率優先で圧縮されている 2 = BLOBは速度優先で圧縮されている
expandedSize	倍長整数	←	非圧縮時のBLOBのサイズ (バイト単位)
currentSize	倍長整数	←	BLOBの現在のサイズ (バイト単位)

説明

BLOB PROPERTIESコマンドは、BLOB blobに関する情報を返します。

compressed引数はBLOBが圧縮されたかどうかを示す値を返します。この値をBLOBテーマの以下の定義済定数と比較できます。

定数	型	値	コメント
Compact compression mode	倍長整数	1	圧縮解凍の処理速度と引き換えに、BLOBをできるだけ小さく圧縮します。デフォルトモード。
Fast compression mode	倍長整数	2	圧縮率と引き換えにBLOBをできるだけ速く圧縮・解凍します (圧縮されたBLOBのサイズは大きくなります)。
GZIP Best compression mode	倍長整数	-1	GZIP圧縮で圧縮率を優先します。
GZIP Fast compression mode	倍長整数	-2	GZIP圧縮で速度を優先します。
Is not compressed	倍長整数	0	

BLOBの圧縮ステータスにかかわらず、expandedSize引数は、圧縮されていない時のBLOBのサイズを戻します。

currentSize引数は、BLOBの現在のサイズを戻します。BLOBが圧縮されている場合には、通常、expandedSizeより小さいcurrentSizeを取得します。BLOBが圧縮されていない場合には、常に、expandedSizeに等しいcurrentSizeを取得します。

例題 1

COMPRESS BLOBおよびEXPAND BLOBの例を参照。

例題 2

BLOBが圧縮された後、以下のプロジェクトメソッドは圧縮できたメモリ空間の割合を返します:

```
  ` Space saved by compression プロジェクトメソッド
  ` Space saved by compression (Pointer {; Pointer } ) -> Long
  ` Space saved by compression ( -> BLOB {; -> savedBytes } ) -> Percentage

 C_POINTER($1;$2)
 C_LONGINT($0;$vlCompressed;$vlExpandedSize;$vlCurrentSize)

 BLOB PROPERTIES($1->;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
 If($vlExpandedSize=0)
    $0:=0
    If(Count parameters>=2)
       $2->:=0
    End if
 Else
    $0:=100-(($vlCurrentSize/$vlExpandedSize)*100)
    If(Count parameters>=2)
       $2->:=$vlExpandedSize-$vlCurrentSize
    End if
 End if
```

このメソッドがアプリケーションに追加された後は、これを以下のように使用できます:

```
  ` ...
 COMPRESS BLOB(vxBlob)
 $vlPercent:=Space saved by compression(->vxBlob;->vlBlobSize)
 ALERT("The compression saved "+String(vlBlobSize)+" bytes, so "+String($vlPercent;"#0%")+
 " of space.")
```

BLOB size

BLOB size (BLOB) -> 戻り値

引数	型		説明
BLOB	BLOB	→	BLOBフィールドまたは変数
戻り値	倍長整数	↩	バイト単位のBLOBサイズ

説明

BLOB size は、blobのサイズをバイト単位で返します。

例題

以下の例はmyBlob BLOBに100バイトを追加します:

```
SET BLOB SIZE(BLOB size(myBlob)+100)
```

BLOB TO DOCUMENT

BLOB TO DOCUMENT (document ; BLOB {; *})

引数	型	説明
document	文字	⇒ドキュメント名
BLOB	BLOB	⇒ドキュメントの新しいコンテンツ
*	演算子	⇒Macintoshのみ: *が渡されれば、リソースフォークに書かれる; そうでなければデータフォークに書かれる

説明

BLOB TO DOCUMENT は、blobに格納されているデータを使用してdocumentの内容全体を上書きします。documentにはドキュメント名を渡します。documentが存在しない場合、コマンドはドキュメントを作成します。既存のドキュメント名を渡す場合、それが開かれていないことを確認してください。ファイルが開かれているとエラーが生成されます。ドキュメントをユーザが選択できるようにするには、Open documentまたはCreate document、およびプロセス変数documentを使用します（例を参照）。

Macintoshでの注意点:

- Macintoshのドキュメントファイルは、データフォークおよびリソースフォークの2つから構成されていることがあります。デフォルトで、BLOB TO DOCUMENTコマンドはドキュメントファイルのデータフォークに書き込みます。リソースフォークに書き込むには、オプション引数 * を渡します。Windowsでは、オプション引数 * は無視されます。
- このコマンドが生成するドキュメントは"タイプ"を持ちません。ドキュメントにタイプを指定するには、SET DOCUMENT TYPE コマンドを使用します。

例題

以下の例は、ドキュメントファイルをすばやく格納、または取得できるような情報システムを記述する場合を想定します。データ入力フォームで、BLOBフィールドにロードされているデータが含まれているドキュメントファイルを保存できるようなボタンを作成します。このボタンのメソッドは、以下の ように作成します:

```
$vhDocRef:=Create document("") ` ドキュメントを作成する
If(OK=1) ` ドキュメントが作成されたら
   CLOSE DOCUMENT($vhDocRef) ` それを閉じる
   BLOB TO DOCUMENT(Document;[YourTable]YourBLOBField) ` ドキュメントに書き込む
   If(OK=0)
 ` エラーを処理する
   End if
End if
```

システム変数およびセット

ドキュメントが正しく書きこまれたらOK変数は1に設定されます。そうでなければ0に設定され、エラーが生成されます。

エラー処理

- 存在しないドキュメントや、既に他のプロセスやアプリケーションで開かれているドキュメントに書き込みしようとすると、対応するファイルマネージャエラーが生成されます。
- 新しいドキュメントを書き込むためのディスク容量が不足する場合があります。
- ドキュメントを書き込む際に、I／Oエラーが発生する場合があります。

いずれの場合でも、ON ERR CALL割り込みメソッドを使用すれば、このエラーをとらえることができます。

⚙ BLOB to integer

BLOB to integer (BLOB ; byteOrder {; offset}) -> 戻り値

引数	型	説明
BLOB	BLOB	→ 整数値を取り出すBLOB
byteOrder	倍長整数	→ 0 Native byte ordering 1 Macintosh byte ordering 2 PC byte ordering
offset	変数	→ BLOB中のオフセット (バイト単位) ← 読み込み後、新しいオフセット
戻り値	整数	↩ 2バイト整数値

説明

BLOB to integer コマンドは、blob BLOBから2バイトの整数値を読み込み、その値を返します。

byteOrder引数は、読み込む2バイト整数値のバイトオーダーを指定します。4Dが提供する以下の定義済み定数のうち、いずれかを渡します:

定数	型	値
Macintosh byte ordering	倍長整数	1
Native byte ordering	倍長整数	0
PC byte ordering	倍長整数	2

プラットフォーム独立性に関する注意

MacintoshとPCプラットフォーム間でBLOBを交換する場合、このコマンドを使用する際のバイトスワップの管理は開発者に任されています。

offset引数を渡した場合、2バイト整数の値はBLOB内のオフセット（ゼロから開始）から読み込まれます。オプション引数offsetを指定しなかった場合には、BLOBの最初の2バイトが読み込まれます。

Note: 0からBLOBサイズ-2の範囲のオフセット値（バイト単位）を渡す必要があります。この範囲の値を渡さないと、エラー-111が生成されます。

呼び出し後、offset変数は、読み込まれたバイト数分だけインクリメントされます。したがって、同じoffset変数を別のBLOB読み込みコマンドにも使用し、別の値をBLOBから読みだすことができます。

例題

以下の例ではBLOBから、オフセット0x200から開始して整数値を20個読み込んでいます:

```
$vlOffset:=0x200
For($viLoop;0;19)
   $viValue:=BLOB to integer(vxSomeBlob;PC byte ordering;$vlOffset)
 ` $viValueに処理を行う
End for
```

BLOB to list

BLOB to list (BLOB {; offset}) -> 戻り値

引数	型		説明
BLOB	BLOB	⇒	階層リストが格納されたBLOB
offset	倍長整数	⇒	BLOB内のオフセット (バイト単位)
		⇐	読み込み後の新しいオフセット
戻り値	ListRef	↩	新しく作成されたリスト参照

説明

BLOB to list コマンドはblob中、offsetで指定されたバイトオフセット（ゼロから開始）に格納されているデータを使用して新しい階層リストを作成し、このリストのリスト参照番号を返します。

BLOBデータはコマンドと整合性を保っていなければなりません。通常、LIST TO BLOBコマンドを使用して階層リストを格納したBLOBを使用します。

オプション引数offsetを指定しない場合には、階層リストはBLOBの最初から読み込まれます。複数の変数やリストが格納されているBLOBを操作する場合には、offsetにオフセットを格納した数値型変数を渡さなければなりません。呼び出しの前に、この数値型変数を適切なオフセットに設定します。呼び出しの後で、この数値型変数はBLOB内に格納されている次の変数へのオフセットを返します。

呼び出し後、階層リストが正常に作成された場合には、システム変数OKは1に設定されます。階層リストを作成するために必要なメモリがない等の理由で処理が実行できなかった場合には、システム変数OKは0に設定されます。

プラットフォーム独立性に関する注意

BLOB to listやLIST TO BLOBは、BLOBに格納されたリストを処理するために4Dの内部フォーマットを使用します。この利点として、これら二つのコマンドを使用する際、プラットフォーム 間のバイトスワップを気にする必要はありません。言い換えればこれらのコマンドを使用してWindowsで作成されたBLOBをMacintoshで使用 する、あるいはその逆を行うことができます。

例題

この例では、データ入力フォームが画面に表示される前に、このフォームのフォームメソッドがBLOBフィールドからリストを抽出し、データ入力が検証されるとこのリストをBLOBフィールドに再び格納します:

```
  ` [Things To Do];"Input" フォームメソッド

 Case of

    :(Form event=On Load)
       hList:=BLOB to list([Things To Do]Other Crazy Ideas)
       If(OK=0)
          hList:=New list
       End if

    :(Form event=On Unload)
       CLEAR LIST(hList;*)

    :(bValidate=1)
       LIST TO BLOB(hList;[Things To Do]Other Crazy Ideas)

 End case
```

システム変数およびセット

リストが正しく作成されると、OK変数には1が設定されます。そうでなければ0が設定されます。

BLOB to longint

BLOB to longint (BLOB ; byteOrder {; offset}) -> 戻り値

引数	型		説明
BLOB	BLOB	→	倍長整数値を取り出すBLOB
byteOrder	倍長整数	→	0 Native byte ordering 1 Macintosh byte ordering 2 PC byte ordering
offset	変数	→	BLOB中のオフセット (バイト単位)
		←	読み込み後、新しいオフセット
戻り値	倍長整数	↩	4バイト整数値

説明

BLOB to longint コマンドは、blob BLOBから4バイトの整数値を読み込み、その値を返します。

byteOrder引数は、読み込む4バイト整数値のバイトオーダーを指定します。4Dが提供する以下の定義済み定数のうち、いずれかを渡します:

定数	型	値
Macintosh byte ordering	倍長整数	1
Native byte ordering	倍長整数	0
PC byte ordering	倍長整数	2

プラットフォーム独立性に関する注意

MacintoshとPCプラットフォーム間でBLOBを交換する場合、このコマンドを使用する際のバイトスワップの管理は開発者に任されています。

offset引数を渡した場合、4バイト整数の値はBLOB内のオフセット（ゼロから開始）から読み込まれます。オプション引数offsetを指定しなかった場合には、BLOBの最初の4バイトが読み込まれます。

Note: 0からBLOBサイズ-4の範囲のオフセット値（バイト単位）を渡す必要があります。この範囲の値を渡さないと、エラー-111が生成されます。

呼び出し後、offset変数は、読み込まれたバイト数分だけインクリメントされます。したがって、同じoffset変数を別のBLOB読み込みコマンドにも使用し、別の値をBLOBから読みだすことができます。

例題

以下の例ではBLOBから、オフセット0x200から開始して倍長整数値を20個読み込んでいます:

```
$vlOffset:=0x200
For($viLoop;0;19)
   $vlValue:=BLOB to longint(vxSomeBlob;PC byte ordering;$vlOffset)
  ` Do something with $vlValue
End for
```

BLOB to real

BLOB to real (BLOB ; realFormat {; offset}) -> 戻り値

引数	型		説明
BLOB	BLOB	→	実数値を取り出すBLOB
realFormat	倍長整数	→	0 Native real format 1 Extended real format 2 Macintosh Double real format 3 Windows Double real format
offset	変数	→	BLOB中のオフセット (バイト単位)
		←	読み込み後、新しいオフセット
戻り値	実数	↩	実数値

説明

BLOB to real コマンドは、blob BLOBから実数値を読み込み、その値を返します。

realFormat引数は、読み込む実数値の内部フォーマットとバイトオーダーを指定します。4Dが提供する以下の定義済み定数のうち、いずれかを渡します:

定数	型	値
Extended real format	倍長整数	1
Macintosh double real format	倍長整数	2
Native real format	倍長整数	0
PC double real format	倍長整数	3

プラットフォーム独立性に関する注意

MacintoshとPCプラットフォーム間でBLOBを交換する場合、このコマンドを使用する際の実数フォーマットとバイトスワップの管理は開発者に任されています。

offset引数を渡した場合、実数の値はBLOB内のオフセット（ゼロから開始）から読み込まれます。オプション引数offsetを指定しなかった場合には、BLOBの最初の8または10バイトが読み込まれます。

Note: 0からBLOBサイズ-8または-10の範囲のオフセット値（バイト単位）を渡す必要があります。この範囲の値を渡さないと、エラー-111が生成されます。

呼び出し後、offset変数は、読み込まれたバイト数分だけインクリメントされます。したがって、同じoffset変数を別のBLOB読み込みコマンドにも使用し、別の値をBLOBから読みだすことができます。

例題

以下の例ではBLOBから、オフセット0x200から開始して実数値を20個読み込んでいます:

```
$vlOffset:=0x200
For($viLoop;0;19)
   $vrValue:=BLOB to real(vxSomeBlob;PC byte ordering;$vlOffset)
  ` Do something with $vrValue
End for
```

BLOB to text

BLOB to text (BLOB ; textFormat {; offset {; textLength}}) -> 戻り値

引数	型		説明
BLOB	BLOB	→	テキストを取り出すBLOB
textFormat	倍長整数	→	テキストのフォーマットと文字セット
offset	変数	→	BLOB内のオフセット (バイト単位)
		←	読み込み後の新しいオフセット
textLength	倍長整数	→	読み込む文字数
戻り値	テキスト	↩	取り出したテキスト

説明

BLOB to text コマンドはBLOB blobからテキストを読みだして、返します。

textFormat 引数は、読み込むテキスト値の内部フォーマットと文字セットを指定します。4Dバージョン11以降で作成されたデータベースでは、4Dはテキストの処理にデフォルトでUnicode (UTF-8) を使用します。互換性を保つため、このコマンドに、以前のバージョンの4Dで使用されていたMac Roman文字セットへの変換を強制することもできます。文字セットはtextFormat 引数で指定します。これを行うには、BLOB テーマの以下の定数のうちいずれかをtextFormat 引数に渡します:

定数	型	値
Mac C string	倍長整数	0
Mac Pascal string	倍長整数	1
Mac Text with length	倍長整数	2
Mac Text without length	倍長整数	3
UTF8 C string	倍長整数	4
UTF8 Text with length	倍長整数	5
UTF8 Text without length	倍長整数	6

Notes:

- "UTF8"から始まる定数は、Unicodeモードのアプリケーションでのみ使用できます。
- "Mac"から始まる定数は、32 KBまでのテキストを扱えます。
- UTF-8以外の文字セットを使用するには、Convert to text コマンドを使用します。

これらの定数とフォーマットに関する詳細は、TEXT TO BLOB コマンドの説明を参照してください。

警告: 読み込む文字数はtextFormat 引数により決定されます。ただしMac Text without lengthとUTF8 Text without lengthは除きます。これらの場合、引数 textLengthで読み込む文字数を指定しなければなりません。他のフォーマットでは、textLengthは無視され、省略できます。

オプションのoffset変数引数を渡すと、テキスト値は0から始まるオフセット位置から読み込まれます。offset変数引数を指定しないと、BLOBの先頭からtextFormatに渡した値に基づき読み込まれます。文字長が指定されないテキストを読み込む際には、offset変数引数を渡さなければならないことに注意してください。

Note: オフセット値として、0 (ゼロ) からBLOBサイズ-テキストサイズの間の数値を渡さなければならないことに注意してください。そうでなければ戻り値は予期できないものとなります。

呼び出し後、offset変数は、読み込まれたバイト数分だけインクリメントされます。したがって、同じoffset変数を別のBLOB読み込みコマンドにも使用し、別の値をBLOBから読みだすことができます。

BLOB TO VARIABLE

BLOB TO VARIABLE (BLOB ; variable {; offset})

引数	型		説明
BLOB	BLOB	→	4D変数を格納したBLOB
variable	変数	←	BLOBの内容を書き込む変数
offset	倍長整数	→	BLOB内の変数の位置
		←	BLOB内の次の変数の位置

説明

BLOB TO VARIABLE コマンドは、offset引数で指定されたバイトオフセット（ゼロから開始）にあるblobに格納されているデータを使用してvariableを上書きします。

BLOBデータは宛先変数と整合性を保っていなければなりません。通常、VARIABLE TO BLOBコマンドを使用して格納されたBLOBを使用します。

オプション引数のoffsetを指定しない場合には、変数データはBLOBの最初から読み込まれます。
複数の変数が格納されているBLOBを操作する場合には、offsetに数値型変数を渡さなければなりません。呼び出しの前に、この数値型変数に適切なオフセットを設定します。呼び出しの後で、この同じ数値型変数はBLOB内に格納されている次の変数へのオフセットを返します。

Note: BLOB TO VARIABLE は C_OBJECT 型のオブジェクト変数をサポートします。詳細な情報に関しては、VARIABLE TO BLOB コマンドを参照して下さい。

呼び出し後、変数が正常に上書きされた場合には、システム変数OKは1に設定されます。変数を上書きするために必要なメモリがない等の理由で処理が実行できなかった場合には、システム変数OKは0に設定されます。

プラットフォーム独立性に関する注意

BLOB TO VARIABLEとVARIABLE TO BLOBは4Dの内部フォーマットを使用してBLOBに格納される変数を取り扱います。この利点として、これら二つのコマンドを使用する際、プラットフォーム間のバイトスワップを気にする必要はありません。言い換えればこれらのコマンドを使用してWindowsで作成されたBLOBをMacintoshで使用する、あるいはその逆を行うことができます。

例題

VARIABLE TO BLOBの例題参照

システム変数およびセット

変数に書き込みが行われると、OK変数に1が設定されます。そうでなければ0が設定されます。

COMPRESS BLOB

COMPRESS BLOB (BLOB {; compression})

引数	型		説明
BLOB	BLOB	→	圧縮するBLOB
compression	倍長整数	→	省略されない場合: 1, 圧縮率優先で圧縮 2, 速度優先で圧縮

説明

COMPRESS BLOBコマンドは、圧縮アルゴリズムを使用してBLOB blobを圧縮します。

オプションのcompression引数を使用すると、BLOBを圧縮する方法を設定できます。BLOBテーマの以下の定義済み定数を指定できます:

定数	型	値	コメント
Compact compression mode	倍長整数	1	圧縮解凍の処理速度と引き換えに、BLOBをできるだけ小さく圧縮します。デフォルトモード。
Fast compression mode	倍長整数	2	圧縮率と引き換えにBLOBをできるだけ速く圧縮・解凍します (圧縮されたBLOBのサイズは大きくなります)。
GZIP Best compression mode	倍長整数	-1	GZIP圧縮で圧縮率を優先します。
GZIP Fast compression mode	倍長整数	-2	GZIP圧縮で速度を優先します。

他の値を渡した場合、またはcompressionを省略した場合、圧縮モード1 (compact internal compression) が使用されます。

注: このコマンドは、サイズが255バイト以上のBLOBだけを圧縮します。

呼び出し後、BLOBが圧縮されればOK変数に1が設定されます。メモリ不足やBLOBサイズが255バイト未満などの理由で圧縮が行われなかった場合、OK変数に0が設定されます。エラーは生成されず、メソッドは実行をレジュームします。
(BLOBが壊れているなど) 他のケースの場合、エラー-10600が生成されます。このエラーはON ERR CALLコマンドを使用してとらえることができます。

BLOB圧縮後、EXPAND BLOBコマンドを使用して解凍できます。

BLOBが圧縮されているかどうかを知るにはBLOB PROPERTIESコマンドを使用します。

警告: 圧縮されたBLOBもBLOBであり、そのコンテンツを編集できます。しかしそうしてしまうと、EXPAND BLOBコマンドは正しくBLOBを解凍できなくなります。

例題 1

この例題 はBLOB vxMyBlobが圧縮されているかテストし、されていなければ圧縮します:

```
BLOB PROPERTIES(vxMyBlob;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
If($vlCompressed=Is not compressed)
   COMPRESS BLOB(vxMyBlob)
End if
```

すでに圧縮されているBLOBにCOMPRESS BLOBを適用した場合、コマンドはそれを検知し、何も行いません。

例題 2

この例題は、ドキュメントを選択させ、それを圧縮します:

```
$vhDocRef :=Open document("")
If(OK=1)
   CLOSE DOCUMENT($vhDocRef)
   DOCUMENT TO BLOB(Document;vxBlob)
   If(OK=1)
      COMPRESS BLOB(vxBlob)
      If(OK=1)
         BLOB TO DOCUMENT(Document;vxBlob)
      End if
   End if
End if
```

例題 3

GZIPで圧縮されたraw HTTPデータを送信します:

```
COMPRESS BLOB($blob;GZIP Best compression mode)
C_TEXT($vEncoding)
$vEncoding:="Content-encoding: gzip"
WEB SET HTTP HEADER($vEncoding)
WEB SEND RAW DATA($blob ;*)
```

システム変数およびセット

BLOBが正しく圧縮されると、システム変数OKは1に設定されます。そうでなければ0に設定されます。

COPY BLOB

COPY BLOB (srcBLOB ; dstBLOB ; srcOffset ; dstOffset ; len)

引数	型		説明
srcBLOB	BLOB	→	コピー元BLOB
dstBLOB	BLOB	→	コピー先BLOB
srcOffset	倍長整数	→	コピー元のコピー開始位置
dstOffset	倍長整数	→	コピー先のコピー開始位置
len	倍長整数	→	コピーするバイト数

説明

COPY BLOB コマンドはlenで指定された数のバイトを、srcBLOBからdstBLOBにコピーします。

コピーはsrcOffsetで指定された場所 (コピー元BLOBの先頭からの相対位置) から開始され、dstOffsetで指定された場所 (コピー先BLOBの先頭からの相対位置) に置かれます。

Note: コピー先BLOBは必要に応じてリサイズされます。

DECRYPT BLOB

DECRYPT BLOB (toDecrypt ; sendPubKey {; recipPrivKey})

引数	型		説明
toDecrypt	BLOB	→	復号するデータ
		←	複合されたデータ
sendPubKey	BLOB	→	送信者の公開鍵
recipPrivKey	BLOB	→	受信者の秘密鍵

説明

DECRYPT BLOB コマンドは、BLOB toDecryptの内容を送信者の公開鍵sendPubKeyを使用して解読します。オプションとして、受信者の秘密鍵recipPrivKeyも使用します。

送信者の公開鍵を納めたBLOBを引数sendPubKeyに渡します。この鍵は、送信者がGENERATE ENCRYPTION KEYPAIRコマンドを実行して生成し、受信者に送信しておく必要があります。

受信者の秘密鍵を納めたBLOBを引数recipPrivKeyに渡すことができます。この引数を渡す場合、受信者はGENERATE ENCRYPTION KEYPAIRコマンドを実行して一組の暗号化鍵を生成し、公開鍵を送信者に送っておく必要があります。この一組の鍵をもとにした暗号化システムでは、送信者のみがメッセージの暗号化を行い、かつ受信者だけがその復号を行えるということが保証されます。一組の鍵をもとにした暗号化システムに関する詳細は、ENCRYPT BLOBコマンドを参照してください。

DECRYPT BLOBコマンドは、（意図的かどうかに関わらず）BLOB内容の変更を防ぐため、チェックサム機能が提供されています。暗号化したBLOBが損傷したり変更されていると、このコマンドは何も行わず、エラーを返します。

例題

ENCRYPT BLOB コマンドの例題を参照

DELETE FROM BLOB

DELETE FROM BLOB (BLOB ; offset ; len)

引数	型	説明
BLOB	BLOB	→バイト列を削除するBLOB
offset	倍長整数	→バイト削除開始位置
len	倍長整数	→削除するバイト数

説明

DELETE FROM BLOB コマンドは、lenで指定されたバイト数を、blobのoffsetで指定された位置（BLOBの最初から相対的な位置として表される）から削除します。BLOBはlenで指定されたバイト数分だけ縮小されます。

DOCUMENT TO BLOB

DOCUMENT TO BLOB (document ; BLOB {; *})

引数	型	説明
document	文字	⇒ドキュメント名
BLOB	BLOB	⇒ドキュメントを受け取るBLOBフィールドまたは変数 ⇐ドキュメントの内容
*	演算子	⇒Macintoshのみ: * が渡されればリソースフォークをロード そうでなければデータフォークをロード

説明

DOCUMENT TO BLOB は、documentの内容全体をblobにロードします。開かれていない既存のドキュメントを渡さなければなりません。そうでなければエラーが生成されます。ドキュメントをユーザが選択できるようにするには、Open documentおよびプロセス変数documentを使用します（例を参照）。

Macintoshでの注意点

Macintoshのドキュメントファイルは、データフォークおよびリソースフォークの2つから構成されていることがあります。デフォルトで、DOCUMENT TO BLOBコマンドはドキュメントファイルのデータフォークを読み込みます。リソースフォークを読み込むには、オプション引数 * を渡します。Windowsでは、オプション引数 * は無視されます。
4D 環境はWindows上でMac OS リソースフォークと同様のものを提供していることに留意してください。例えば、4Dデータベースのデータフォークは拡張子.4DBのファイルに格納され、リソースフォークは同じ名前の拡張子.RSRのファイルに格納されます。Windowsでは、BLOBに格納されるデータフォークやリソースフォークを扱 うには、フォークに対応するファイルにアクセスします。

例題

以下の例は、ドキュメントファイルをすばやく格納、または取得できるような情報システムを記述する場合を想定します。データ入力フォームで、ドキュメントファイルをBLOBフィールドにロードできるようなボタンを作成します。このボタンに以下のようなメソッドを作成します:

```
$vhDocRef:=Open document("") ` ドキュメントを選択させる
If(OK=1) ` ドキュメントが選択されたら
   CLOSE DOCUMENT($vhDocRef) ` それを閉じる
   DOCUMENT TO BLOB(Document;[YourTable]YourBLOBField) ` ドキュメントをロード
   If(OK=0)
  ` エラー処理
   End if
End if
```

システム変数およびセット

ドキュメントが正しくロードされると、OK変数は1に設定されます。そうでなければ0に設定され、エラーが生成されます。

エラー処理

- 存在しないドキュメントファイルや、既に他のプロセスやアプリケーションで開かれているドキュメントファイルを（BLOBに）ロードしようとすると、それぞれに対応するファイルマネージャエラーが生成されます。
- ドキュメントファイルがロックされていたり、ロックされているボリュームにあったり、ドキュメントファイルを読み込む際に問題が発生すると、I／Oエラーが発生する場合があります。
- メモリ不足のためにドキュメントファイルをロードできない場合には、エラー-108が生成されます。

いずれの場合でも、ON ERR CALL割り込みメソッドを使用すれば、このエラーをとらえることができます。

ENCRYPT BLOB

ENCRYPT BLOB (toEncrypt ; sendPrivKey {; recipPubKey})

引数	型		説明
toEncrypt	BLOB	→	暗号化するデータ
		←	暗号化されたデータ
sendPrivKey	BLOB	→	送信者の秘密鍵
recipPubKey	BLOB	→	受信者の公開鍵

説明

ENCRYPT BLOBコマンドは、toEncrypt BLOBの内容を送信者の秘密鍵sendPrivKeyを使用して暗号化します。オプションとして、同時に受信者の公開鍵recipPubKeyも使用できます。これらの鍵は、GENERATE ENCRYPTION KEYPAIR（セキュアプロトコルテーマ）を使用して生成します。

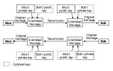
Note: このコマンドは、SSLプロトコルアルゴリズムおよび暗号化機能を利用します。したがって、このコマンドを使用するためには、4D Webサーバ通信にSSLを使用しない場合でも、SSLプロトコルに必要となる各コンポーネントがマシン上に正しくインストールされているか確認してください。このプロトコルについての詳細は、の節を参照してください。

- 送信者の秘密鍵のみを暗号化に使用されると、公開鍵の所有者だけがこの情報を読み取れます。このシステムにより、送信者自身が情報の暗号化を行ったということが保証されます。
- 送信者の秘密鍵と受信者の公開鍵を同時に使用することにより、情報の読み取りを行えるのは1人の受信者だけであることが保証されます。

鍵を納めるBLOBは、PKCS内部フォーマットです。クロスプラットフォームであるこの形式では、電子メールやテキストファイルにコピー＆ペーストすることにより簡単に鍵のやり取りや処理を行うことができます。

コマンドを実行すると、BLOB toEncryptには暗号化されたデータが納められます。このデータの解読は、引数として渡された送信者の公開鍵を使用した上でDECRYPT BLOBコマンドによってのみ行えます。さらに、情報の暗号化の際にオプションである受信者の公開鍵を使用すると、解読には受信者の秘密鍵も必要になります。

“Alice”と“Bob”の間で行われるメッセージ交換のための、公開及び秘密鍵を使用した暗号化の原則



Note: BLOB内容の変更（意図的かどうかに関わらず）を防ぐため、暗号にはチェックサム機能が含まれています。したがって、暗号化されたBLOBは変更しないでください。変更を行うと、解読できなくなるおそれがあります。

暗号化コマンドの最適化

データの暗号化を行うと、アプリケーションの実行速度が低下し、2つの鍵を使用した場合は特に遅くなります。しかし、以下の最適化に関するヒントを考慮してみることをお勧めします:

- 現時点で使用可能なメモリに応じて、コマンドは“同期”モードまたは“非同期”モードで実行されます。
  非同期モードでは他のプロセスを中断しないので、より高速になります。使用可能なメモリが、暗号化するデータの少なくとも2倍ある場合、このモードが自動的に使用されます。
  メモリがそれ以下の場合、セキュリティ上の理由から、同期モードが使用されます。このモードでは他のプロセスを中断するため、速度はより低下します。
- BLOBのサイズが大きい場合、BLOBの重要な小さな部分のみを暗号化して、処理しなければならないデータ量や時間を減らすことができます。

例題

一つのキーを使用する

会社で、4Dデータベースに格納されるデータを秘密にしておきたいとします。そしてこれらの情報をファイルにしてインターネット経由で子会社に送信する必要があります。

1) 会社はGENERATE ENCRYPTION KEYPAIRコマンドを使用して一組の鍵を生成します:

```
  `Method GENERATE_KEYS_TXT
 C_BLOB($BPublicKey;$BPrivateKey)
 GENERATE ENCRYPTION KEYPAIR($BPrivateKey;$BPublicKey)
 BLOB TO DOCUMENT("PublicKey.txt";$BPublicKey)
 BLOB TO DOCUMENT("PrivateKey.txt";$BPrivateKey)
```

2) 会社側は秘密鍵を保存し、各子会社へは公開鍵を含むドキュメントのコピーを送信します。最高のセキュリティを維持するため、鍵は子会社に手渡すディスクにコピーしてください。

3) 次に、機密情報（例えば、テキストフィールドに保存したもの）をBLOBにコピーします。この情報は、秘密鍵を使用して暗号化されます:

```
  `Method ENCRYPT_INFO
 C_BLOB($vbEncrypted;$vbPrivateKey)
 C_TEXT($vtEncrypted)
 $vtEncrypted:=[Private]Info
 VARIABLE TO BLOB($vtEncrypted;$vbEncrypted)
 DOCUMENT TO BLOB("PrivateKey.txt";$vbPrivateKey)
 If(OK=1)
    ENCRYPT BLOB($vbEncrypted;$vbPrivateKey)
    BLOB TO DOCUMENT("Update.txt";$vbEncrypted)
 End if
```

4) 更新用ファイルは、インターネットのような非暗号化チャネルで子会社に送ることができます。万一、第三者がこの暗号化されたファイルを入手した場合でも、公開鍵なしではファイルを解読できません。

5) 各子会社では、公開鍵を使用してドキュメントの解読が可能です:

```
  `Method DECRYPT_INFO
 C_BLOB($vbEncrypted;$vbPublicKey)
 C_TEXT($vtDecrytped)
 C_TIME($vtDocRef)
 ALERT("Please select an encrypted document.")
 $vtDocRef:=Open document("") `Select Update.txt
 If(OK=1)
    CLOSE DOCUMENT($vtDocRef)
    DOCUMENT TO BLOB(Document;$vbEncrypted)
    DOCUMENT TO BLOB("PublicKey.txt";$vbPublicKey)
    If(OK=1)
       DECRYPT BLOB($vbEncrypted;$vbPublicKey)
       BLOB TO VARIABLE($vbEncrypted;$vtDecrypted)
       CREATE RECORD([Private])
       [Private]Info:=$vtDecrypted
       SAVE RECORD([Private])
    End if
 End if
```

キーペアを使用

ある会社が情報のやり取りにインターネットを利用したいものとします。各子会社は機密情報を受信し、また本社へ情報の送信も行います。したがって要件は次の2 つです:
- 受信だけがメッセージを読むことができます。
- 受信側は、メッセージの送信が送信者自身によって行われたという証拠を取得しなければなりません。

1) 本社および各子会社では、それぞれ独自の鍵のペアを生成します（GENERATE_KEYS_TXTGENERATE_KEYS_TXTメソッドを使用）。

2) 秘密鍵は双方で秘密にしておきます。各子会社は、自分の公開鍵を本社へ送り、本社もまた独自の公開鍵を送信します。公開鍵ではメッセージを解読するのに十分ではないため、この鍵の転送に暗号化のチャネルを使用する必要はありません。

3) 送信する情報を暗号化するため、子会社や本社ではENCRYPT_INFO_2メソッドを実行します。このメソッドは、送信側の秘密鍵と受信側の公開鍵を使用して情報の暗号化を行います:

```
  `Method ENCRYPT_INFO_2
 C_BLOB($vbEncrypted;$vbPrivateKey;$vbPublicKey)
 C_TEXT($vtEncrypt)C_TIME($vtDocRef)
 $vtEncrypt:=[Private]Info
 VARIABLE TO BLOB($vtEncrypt;$vbEncrypted)
  ` Your own private key is loaded...
 DOCUMENT TO BLOB("PrivateKey.txt";$vbPrivateKey)
 If(OK=1)
  ` ...and the recipient's public key
    ALERT("Please select the recipient's public key.")
    $vhDocRef:=Open document("") `Public key to load
    If(OK=1)
       CLOSE DOCUMENT($vtDocRef)
       DOCUMENT TO BLOB(Document;$vbPublicKey)
  `BLOB encryption with the two keys as parameters
       ENCRYPT BLOB($vbEncrypted;$vbPrivateKey;$vbPublicKey)
       BLOB TO DOCUMENT("Update.txt";$vbEncrypted)
    End if
 End if
```

4) 暗号化したファイルが、インターネット経由で受信側に送信されます。万が一第三者がこのファイルを入手した場合、たとえ公開鍵を持っていたとしても、受信側の秘密鍵も必要となるため、メッセージを解読できません。

5) 受信側はそれぞれ、独自の秘密鍵と送信側の公開鍵を使用してドキュメントの解読が可能です:

```
  `Method DECRYPT_INFO_2
 C_BLOB($vbEncrypted;$vbPublicKey;$vbPrivateKey)
 C_TEXT($vtDecrypted)
 C_TIME($vhDocRef)
 ALERT("Please select the encrypted document.")
 $vhDocRef:=Open document("") `Select the Update.txt file
 If(OK=1)
    CLOSE DOCUMENT($vhDocRef)
    DOCUMENT TO BLOB(Document;$vbEncrypted)
  `Your own private key is loaded
    DOCUMENT TO BLOB("PrivateKey.txt";$vbPrivateKey)
    If(OK=1)
  ` ...and the sender's public key
       ALERT("Please select the sender's public key.")
       $vhDocRef:=Open document("") `Public key to load
       If(OK=1)
          CLOSE DOCUMENT($vhDocRef)
          DOCUMENT TO BLOB(Document;$vbPublicKey)
  `Decrypting the BLOB with two keys as parameters
          DECRYPT BLOB($vbEncrypted;$vbPublicKey;$vbPrivateKey)
          BLOB TO VARIABLE($vbEncrypted;$vtDecrypted)
          CREATE RECORD([Private])
          [Private]Info:=$vtDecrypted
          SAVE RECORD([Private])
       End if
    End if
 End if
```

EXPAND BLOB

EXPAND BLOB (BLOB)

引数	型		説明
BLOB	BLOB	→	展開するBLOB

説明

EXPAND BLOB コマンドは、COMPRESS BLOBコマンドを使用して既に圧縮されているblobを解凍します。

呼び出し後、BLOBが解凍された場合は、システム変数OKは1に設定されます。BLOBが解凍できなかった場合は、システム変数OKは0に設定されます。

メモリ不足で解凍できない場合は、エラーが表示されず、メソッド実行をレジュームします。
(BLOBが圧縮されていなかったり壊れていたりするなど) その他の場合、-10600のエラーを生成します。このエラーは、ON ERR CALLコマンドでとらえることができます。

BLOBが圧縮されているかを検証するには、BLOB PROPERTIES コマンドを使用します。

例題 1

この例題はBLOB vxMyBlob が圧縮されているかテストし、圧縮されていれば解凍します:

```
BLOB PROPERTIES(vxMyBlob;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
If($vlCompressed#Is not compressed)
   EXPAND BLOB(vxMyBlob)
End if
```

例題 2

この例題はドキュメントを選択させ、圧縮されていれば解凍します:

```
$vhDocRef :=Open document("")
If(OK=1)
   CLOSE DOCUMENT($vhDocRef)
   DOCUMENT TO BLOB(Document;vxBlob)
   If(OK=1)
      BLOB PROPERTIES(vxBlob;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
      If($vlCompressed#Is not compressed)
         EXPAND BLOB(vxBlob)
         If(OK=1)
            BLOB TO DOCUMENT(Document;vxBlob)
         End if
      End if
   End if
End if
```

システム変数およびセット

BLOBが正しく解凍されるとOK変数は1に、そうでなければ0に設定されます。

INSERT IN BLOB

INSERT IN BLOB (BLOB ; offset ; len {; filler})

引数	型		説明
BLOB	BLOB	→	バイト列を挿入するBLOB
offset	倍長整数	→	バイト列挿入開始位置
len	倍長整数	→	挿入するバイト数
filler	倍長整数	→	デフォルトのバイト値 (0x00..0xFF) 省略した場合0x00

説明

INSERT IN BLOB コマンドは、blobのoffsetで指定した位置に、lenで指定した数のバイトを挿入します。BLOBはlenバイトだけ大きくなります。

オプションのfiller引数を指定しない場合、BLOBに挿入されるバイトは0x00に設定されます。それ以外の場合、fillerに渡した値に設定されます (モジューロ 256 - 0..255)。

呼び出し前に、offset引数にはBLOBの先頭から相対的な挿入位置を設定します。

INTEGER TO BLOB

INTEGER TO BLOB (entier ; BLOB ; ordreOctet {; offset | *})

引数	型	説明
entier	倍長整数	⇒ BLOBに書き込む整数値
BLOB	BLOB	⇒ 整数値を受け取るBLOB
ordreOctet	倍長整数	⇒ 0 Native byte ordering 1 Macintosh byte ordering 2 PC byte ordering
offset | *	変数, 演算子	⇒ BLOB内のオフセット (バイト単位) または値を追加する場合 * ⇐ *でない場合、書き込み後の新しいオフセット

説明

INTEGER TO BLOB コマンドは、blobに2バイトのinteger値を書き込みます。

byteOrder引数は、2バイト整数値が書き込まれる際のバイトオーダーを決定します。4Dが提供する以下の定義済み定数のうち、いずれか1つを渡します:

定数	型	値
Macintosh byte ordering	倍長整数	1
Native byte ordering	倍長整数	0
PC byte ordering	倍長整数	2

プラットフォーム独立性に関する注意

MacintoshとPCプラットフォーム間でBLOBを交換する場合、このコマンドを使用する際のバイトスワップの管理は開発者に任されています。

オプションの * 引数を渡すと、2バイト整数値はBLOBの最後に追加され、BLOBサイズはそれに従い拡張されます。オプションの * 引数を使用することで、BLOBがメモリに収まる限り、整数, 倍長整数, 実数 あるいは テキスト値 (他のBLOBコマンド参照) をBLOBに連続して格納できます。

オプション引数 * やoffset変数引数を指定しない場合、2バイトの整数値はBLOBの最初に格納され、それ以前の内容を上書きします。これに合わせてBLOBのサイズも調整されます。

offset変数引数を渡した場合、2バイト整数値はBLOB内のオフセット（ゼロから開始）に書き込まれます。2バイトの整数の値を書き込む位置に関わらず、BLOBのサイズは渡した位置に従って（必要に応じてさらに最大2バイトまで）増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、offset変数引数は、書き込まれたバイト数分だけインクリメントされます。したがって、同じoffset変数引数を別のBLOB書き込みコマンドにも使用し、別の値をBLOBに追加できます。

例題 1

以下のコードを実行すると:

```
INTEGER TO BLOB(0x0206;vxBlob;Native byte ordering)
```

- vxBlobのサイズは2バイトになります。
- Power PCプラットフォーム: vxBlob{0} = $02 で vxBLOB{1} = $06
- Intelプラットフォーム: vxBLOB{0} = $06 で vxBLOB{1} = $02

例題 2

以下のコードを実行すると:

```
INTEGER TO BLOB(0x0206;vxBlob;Macintosh byte ordering)
```

- vxBlobのサイズは2バイトになります。
- すべてのプラットフォーム: vxBLOB{0} = $02 で vxBLOB{1} = $06

例題 3

以下のコードを実行すると:

```
INTEGER TO BLOB(0x0206;vxBlob;PC byte ordering)
```

- vxBlobのサイズは2バイトになります。
- すべてのプラットフォーム: vxBLOB{0} = $06 で vxBLOB{1} = $02

例題 4

以下のコードを実行すると:

```
SET BLOB SIZE(vxBlob;100)
INTEGER TO BLOB(0x0206;vxBlob;PC byte ordering;*)
```

- vxBlobのサイズは102バイトになります。
- すべてのプラットフォーム: vxBLOB{100} = $06 で vxBLOB{101} = $02
- BLOBの他のバイトは変更されない

例題 5

以下のコードを実行すると:

```
SET BLOB SIZE(vxBlob;100)
vlOffset:=50
INTEGER TO BLOB(0x0206;vxBlob;Macintosh byte ordering;vlOffset)
```

- vxBlobのサイズは100バイトになります。
- すべてのプラットフォーム: vxBLOB{50} = $02 で vxBLOB{51} = $06
- BLOBの他のバイトは変更されない
- 変数vlOffsetは2インクリメントされ52となる

LIST TO BLOB

LIST TO BLOB (list ; BLOB {; *})

引数	型		説明
list	ListRef	→	BLOBに格納する階層リスト
BLOB	BLOB	→	階層リストを受け取るBLOB
*	演算子	→	値を追加するには*

説明

LIST TO BLOB コマンドは、blobに階層リストlistを格納します。

オプション引数 * を指定した場合、階層リストはBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数 * を使用すれば、BLOBがメモリに収まる限り、変数やリストを (他のBLOBコマンド参照) をBLOBに連続して格納できます。

オプション引数 * を指定しない場合には、階層リストはBLOBの最初に格納され、それ以前の内容を上書きします。これに合わせてBLOBのサイズも調整されます。

階層リストの格納場所に関わらず、指定された位置に従ってBLOBのサイズは必要に応じて拡張されます（必要な場合にはリストサイズまで加算）。修正後のバイトは（設定されたもの以外）0（ゼロ）にリセットされます。

警告: BLOBを使用して階層リストを格納すると、階層リストは4D内部形式を使用してBLOBに格納されるため、格納されたBLOBの内容を読み出すにはBLOB to listを使用しなければなりません。

呼び出し後、階層リストが正常に格納された場合には、システム変数OKは1に設定されます。階層リストを格納するために必要なメモリがない等の理由で処理が実行できなかった場合には、システム変数OKは0に設定されます。

プラットフォーム独立性に関する注意

LIST TO BLOBやBLOB to listは、BLOBに格納されたリストを処理するために4Dの内部フォーマットを使用します。この利点として、これら二つのコマンドを使用する際、プラットフォーム 間のバイトスワップを気にする必要はありません。言い換えればこれらのコマンドを使用してWindowsで作成されたBLOBをMacintoshで使用 する、あるいはその逆を行うことができます。

例題

BLOB to listの例題参照

LONGINT TO BLOB

LONGINT TO BLOB (longint ; BLOB ; byteOrder {; offset | *})

引数	型	説明
longint	倍長整数	→ BLOBに書き込む倍長整数値
BLOB	BLOB	→ 倍長整数値を受け取るBLOB
byteOrder	倍長整数	→ 0 Native byte ordering 1 Macintosh byte ordering 2 PC byte ordering
offset | *	変数, 演算子	→ BLOB内のオフセット (バイト単位) または値を追加する場合 * ← *でない場合、書き込み後の新しいオフセット

説明

LONGINT TO BLOB コマンドは、blobに4バイトのlongint値を書き込みます。

byteOrder引数は、4バイト倍長整数値が書き込まれる際のバイトオーダーを決定します。4Dが提供する以下の定義済み定数のうち、いずれか1つを渡します:

定数	型	値
Macintosh byte ordering	倍長整数	1
Native byte ordering	倍長整数	0
PC byte ordering	倍長整数	2

プラットフォーム独立性に関する注意

MacintoshとPCプラットフォーム間でBLOBを交換する場合、このコマンドを使用する際のバイトスワップの管理は開発者に任されています。

オプションの * 引数を渡すと、4バイト倍長整数値はBLOBの最後に追加され、BLOBサイズはそれに従い拡張されます。オプションの * 引数を使用することで、BLOBがメモリに収まる限り、整数, 倍長整数, 実数 あるいは テキスト値 (他のBLOBコマンド参照) をBLOBに連続して格納できます。

オプション引数 * やoffset変数引数を指定しない場合、4バイトの倍長整数値はBLOBの最初に格納され、それ以前の内容を上書きします。これに合わせてBLOBのサイズも調整されます。

offset変数引数を渡した場合、4バイト倍長整数値はBLOB内のオフセット (ゼロから開始) に書き込まれます。4バイトの倍長整数の値を書き込む位置に関わらず、BLOBのサイズは渡した位置に従って (必要に応じてさらに最大4 バイトまで) 増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、offset変数引数は、書き込まれたバイト数分だけインクリメントされます。したがって、同じoffset変数引数を別のBLOB書き込みコマンドにも使用し、別の値をBLOBに追加できます。

例題 1

以下のコードを実行すると:

```
LONGINT TO BLOB(0x01020304;vxBlob;Native byte ordering)
```

- vxBlobのサイズは4バイトになります。
- Power PCプラットフォーム: vxBLOB{0}=$01, vxBLOB{1}=$02, vxBLOB{2}=$03, vxBLOB{3}=$04
- Intelプラットフォーム: vxBLOB{0}=$04, vxBLOB{1}=$03, vxBLOB{2}=$02, vxBLOB{3}=$01

例題 2

以下のコードを実行すると:

```
LONGINT TO BLOB(0x01020304;vxBlob;Macintosh byte ordering)
```

- vxBlobのサイズは4バイトになります。
- すべてのプラットフォーム: vxBLOB{0}=$01, vxBLOB{1}=$02, vxBLOB{2}=$03, vxBLOB{3}=$04

例題 3

以下のコードを実行すると:

```
LONGINT TO BLOB(0x01020304;vxBlob;PC byte ordering)
```

- vxBlobのサイズは4バイトになります。
- すべてのプラットフォーム: vxBLOB{0}=$04, vxBLOB{1}=$03, vxBLOB{2}=$02, vxBLOB{3}=$01

例題 4

以下のコードを実行すると:

```
SET BLOB SIZE(vxBlob;100)
LONGINT TO BLOB(0x01020304;vxBlob;PC byte ordering;*)
```

- vxBlobのサイズは104バイトになります。
- すべてのプラットフォーム: vxBLOB{100}=$04, vxBLOB{101}=$03, vxBLOB{102}=$02, vxBLOB{103}=$01
- BLOBの他のバイトは変更されない

例題 5

以下のコードを実行すると:

```
SET BLOB SIZE(vxBlob;100)
vlOffset:=50
LONGINT TO BLOB(0x01020304;vxBlob;Macintosh byte ordering;vlOffset)
```

- vxBlobのサイズは100バイトになります。
- すべてのプラットフォーム: vxBLOB{50}=$01, vxBLOB{51}=$02, vxBLOB{52}=$03, vxBLOB{53}=$04
- BLOBの他のバイトは変更されない
- 変数vlOffsetは4インクリメントされ54となる

REAL TO BLOB

REAL TO BLOB (real ; BLOB ; realFormat {; offset | *})

引数	型	説明
real	実数	→ BLOBに書き込む実数値
BLOB	BLOB	→ 実数値を受け取るBLOB
realFormat	倍長整数	→ 0 Native real format 1 Extended real format 2 Macintosh Double real format 3 Windows Double real format
offset | *	変数, 演算子	→ BLOB内のオフセット (バイト単位) または値を追加する場合 * ← *でない場合、書き込み後の新しいオフセット

説明

REAL TO BLOB コマンドは、blobに実数値realを書き込みます。

realFormat引数は、実数値が書き込まれる際の内部フォーマットバイトオーダーを決定します。4Dが提供する以下の定義済み定数のうち、いずれか1つを渡します:

定数	型	値
Extended real format	倍長整数	1
Macintosh double real format	倍長整数	2
Native real format	倍長整数	0
PC double real format	倍長整数	3

プラットフォーム独立性に関する注意

MacintoshとPCプラットフォーム間でBLOBを交換する場合、このコマンドを使用する際の実数フォーマットとバイトスワップの管理は開発者に任されています。

オプションの * 引数を渡すと、実数値はBLOBの最後に追加され、BLOBサイズはそれに従い拡張されます。オプションの * 引数を使用することで、BLOBがメモリに収まる限り、整数, 倍長整数, 実数 あるいは テキスト値 (他のBLOBコマンド参照) をBLOBに連続して格納できます。

オプション引数 * やoffset変数引数を指定しない場合、実数値はBLOBの最初に格納され、それ以前の内容を上書きします。これに合わせてBLOBのサイズも調整されます。

offset変数引数を渡した場合、実数値はBLOB内のオフセット(ゼロから開 始)に書き込まれます。実数値を書き込む位置に関わらず、BLOBのサイズは渡した位置に従って(必要に応じてさらに最大8または10バイトまで)増加し ます。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、offset変数引数は、書き込まれたバイト数分だけインクリメントされます。したがって、同じoffset変数引数を別のBLOB書き込みコマンドにも使用し、別の値をBLOBに追加できます。

例題 1

以下のコードを実行すると:

```
C_REAL(vrValue)
vrValue:=...
REAL TO BLOB(vrValue;vxBlob;Native real format)
```

- すべてのプラットフォームでvxBlobのサイズは8 bytesになります

例題 2

以下のコードを実行すると:

```
C_REAL(vrValue)
vrValue:=...
REAL TO BLOB(vrValue;vxBlob;Extended real format)
```

- すべてのプラットフォームでvxBlobのサイズは10bytesになります

例題 3

以下のコードを実行すると:

```
C_REAL(vrValue)
vrValue:=...
REAL TO BLOB(vrValue;vxBlob;Macintosh double real format) ` または PC double real format
```

- すべてのプラットフォームでvxBlobのサイズは8 bytesになります

例題 4

以下のコードを実行すると:

```
SET BLOB SIZE(vxBlob;100)
C_REAL(vrValue)
vrValue:=...
INTEGER TO BLOB(vrValue;vxBlob;Windows Double real format) ` または Macintosh double real format
```

すべてのプラットフォームでvxBlobのサイズは8 bytesになります

例題 5

以下のコードを実行すると:

```
SET BLOB SIZE(vxBlob;100)
REAL TO BLOB(vrValue;vxBlob;Extended real format;*)
```

- すべてのプラットフォームでvxBlobのサイズは110bytesになります
- すべてのプラットフォームで、実数値は#100から#109までのバイトに書き込まれます
- BLOBの他のバイトは変更されません

例題 6

以下のコードを実行すると:

```
SET BLOB SIZE(vxBlob;100)
C_REAL(vrValue)
vrValue:=...
vlOffset:=50
REAL TO BLOB(vrValue;vxBlob;Windows Double real format;vlOffset) ` または Macintosh double real format
```

- すべてのプラットフォームでvxBlobのサイズは100bytesになります
- すべてのプラットフォームで、実数値は#50から#57までのバイトに書き込まれます
- BLOBの他のバイトは変更されません
- 変数vlOffsetは8インクリメントされ58となります

SET BLOB SIZE

SET BLOB SIZE (BLOB ; size {; filler})

引数	型		説明
BLOB	BLOB	➡	BLOBフィールドまたは変数
size	倍長整数	➡	BLOBの新しいサイズ
filler	倍長整数	➡	埋め込み文字のASCIIコード

説明

SET BLOB SIZE コマンドは、size引数に渡された値に従って、BLOB blobのサイズを変更します。

BLOBに新しいバイトを割り当て、それらのバイトを特定の値で初期化したい場合には、その値 (0..255) をオプション引数のfillerに渡します。

例題 1

大きなBLOBプロセスまたはインタープロセス変数の処理を終了した後、占有していたメモリを解放することをお勧めします。そのためには、以下のように記述します:

```
SET BLOB SIZE(aProcessBLOB;0)
SET BLOB SIZE(<>anInterprocessBLOB;0)
```

例題 2

以下の例では、0xFFが埋め込まれた16KBのBLOBが作成されます:

```
C_BLOB(vxData)
SET BLOB SIZE(vxData;16*1024;0xFF)
```

エラー処理

メモリが足りず、BLOBのリサイズができないとき、エラー-108が生成されます。このエラーはON ERR CALLでとらえることができます。

TEXT TO BLOB

TEXT TO BLOB (text ; BLOB {; textFormat {; offset | *}})

引数	型		説明
text	文字	→	BLOBに書き込むテキスト
BLOB	BLOB	→	テキストを受け取るBLOB
textFormat	倍長整数	→	テキストのフォーマットと文字セット
offset | *	変数, 演算子	→ ←	BLOB内のオフセット (バイト単位) または値を追加する場合 * *でない場合、書き込み後の新しいオフセット

説明

TEXT TO BLOBコマンドはテキスト値 textをBLOB blobに書き込みます。

textFormat 引数を使用して、書き込むテキスト値の内部フォーマットと文字セットを指定できます。これを行うには、BLOBテーマの以下の定数のうちいずれかをtextFormat 引数に渡します:

定数	型	値
Mac C string	倍長整数	0
Mac Pascal string	倍長整数	1
Mac Text with length	倍長整数	2
Mac Text without length	倍長整数	3
UTF8 C string	倍長整数	4
UTF8 Text with length	倍長整数	5
UTF8 Text without length	倍長整数	6

textFormat 引数を省略した場合、デフォルトで4DはMac C Stringフォーマットを使用します。4Dバージョン11以降で作成されたデータベースでは、4Dはテキストの処理にデフォルトでUnicode文字セット (UTF-8) を扱います。そのためこの文字セットを利用が推奨されます。

Notes:

- "UTF8"から始まる定数は、Unicodeモードのアプリケーションでのみ使用できます。
- "Mac"から始まる定数は、32 KBまでのテキストを扱えます。
- UTF-8以外の文字セットを使用するにはCONVERT FROM TEXTコマンドを使用します。

これらのフォーマットについて説明します:

テキストフォーマット	説明と例題	
C string	テキストはNULL文字 (ASCIIコード $00)で終了する	
	UTF8	"" --> $00
		"Café" --> $43 61 66 C3 A9 00
	Mac	"" --> $00
		"Café" --> $43 61 66 8E 00
Pascal string	テキストの前に1バイトのテキスト長が置かれる	
	UTF8	-
		-
	Mac	"" --> $00
		"Café" --> $04 43 61 66 8E
Text with length	テキストの前に4バイト (UTF8) または2バイト (Mac) のテキスト長が置かれる	
	UTF8	"" --> $00 00 00 00
		"Café" --> $00 00 00 05 43 61 66 C3 A9
	Mac	"" --> $00 00
		"Café" --> $00 04 43 61 66 8E
Text without length	テキストはその文字だけで構成される	
	UTF8	"" --> データなし
		"Café" --> $43 61 66 C3 A9
	Mac	"" --> データなし
		"Café" --> $43 61 66 8E

オプションの * 引数を渡すと、テキストはBLOBの最後に追加され、BLOBサイズはそれに従い拡張されます。オプションの * 引数を使用することで、BLOBがメモリに収まる限り、整数, 倍長整数, 実数 あるいは テキスト値 (他のBLOBコマンド参照) をBLOBに連続して格納できます。

オプション引数 * やoffset変数引数を指定しない場合、テキストはBLOBの最初に格納され、それ以前の内容を上書きします。これに合わせてBLOBのサイズも調整されます。

offset変数引数を渡した場合、テキストはBLOB内のオフセット(ゼロから開始)に書き込まれます。テキストを書き込む位置に関わらず、BLOBのサイズは渡した位置に従って(必要に応じてさらにテキストのサイズまで)増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、offset変数引数は、書き込まれたバイト数分だけインクリメントされます。したがって、同じoffset変数引数を別のBLOB書き込みコマンドにも使用し、別の値をBLOBに追加できます。

例題

```
SET BLOB SIZE(vxBlob;0)
C_TEXT(vtValue)
vtValue:="Cafe" ` vtValue長さは4バイト
TEXT TO BLOB(vtValue;vxBlob;Mac C string) ` BLOBのサイズは5 bytes
TEXT TO BLOB(vtValue;vxBlob;Mac Pascal string) ` BLOBのサイズは5 bytes
TEXT TO BLOB(vtValue;vxBlob;Mac Text with length) ` BLOBのサイズは6 bytes
TEXT TO BLOB(vtValue;vxBlob;Mac Text without length) ` BLOBのサイズは4 bytes
TEXT TO BLOB(vtValue;vxBlob;UTF8 C string) ` BLOBのサイズは6 bytes
TEXT TO BLOB(vtValue;vxBlob;UTF8 Text with length) ` BLOBのサイズは9 bytes
TEXT TO BLOB(vtValue;vxBlob;UTF8 Text without length) ` BLOBのサイズは5 bytes
```

VARIABLE TO BLOB

VARIABLE TO BLOB (variable ; BLOB {; offset | *})

引数	型	説明
variable	変数	→BLOBに格納する変数
BLOB	BLOB	→変数を受け取るBLOB
offset | *	変数, 演算子	→BLOB内のオフセット (バイト単位) または値を追加する場合 *
		←*でない場合、書き込み後の新しいオフセット

説明

VARIABLE TO BLOBコマンドは、variableをblobに格納します。

オプション引数 * を指定した場合には、変数はBLOBの最後に追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数 * を使用すれば、BLOBがメモリ容量内であれば変数やリスト（他のBLOBコマンドを参照してください）をいくつでも順番にBLOBの中に格納できます。

オプション引数 * やoffset変数引数を指定しない場合、変数はBLOBの先頭に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

offsetに変数引数を渡すと、変数値のオフセット位置（ゼロから始めます）からBLOBに書き込まれます。変数を書き込む位置にかかわらず、BLOBのサイズは渡した場所に応じて増やされます (必要に応じ変数のサイズも)。新しく割り当てられたバイトは、ゼロに初期化されます。

呼び出し後、offset変数引数は書き込まれたバイト数だけインクリメントされます。その後同じ変数を他のBLOB書き込みコマンドで使用してさらに変数やリストを書き込めます。

VARIABLE TO BLOBコマンドは、以下のものを除いて、どのようなタイプの変数でも（他のBLOBも）受け付けます:

- ポインタ
- ポインタ配列

以下の点に注意して下さい:

- 階層リスト (ListRef) への参照である倍長整数の変数を保存した場合には、**VARIABLE TO BLOB**コマンドは階層リストではなく倍長整数変数を格納します。BLOB内に階層リストを格納、またはBLOBから階層リストを取り出すには、LIST TO BLOB と BLOB to list を使用します。
- C_OBJECT オブジェクトを variable 引数に渡した場合、コマンドはそれをUTF-8のJSON形式でBLOBの中に保存します。オブジェクトがポインターを含んでいた場合、ポインター自身ではなくそれが参照していた値が保存されます。

ただし、階層リスト (ListRef) への参照である倍長整数の変数を格納した場合には、**VARIABLE TO BLOB**コマンドは階層リストではなく倍長整数変数を格納します。BLOB内に階層リストを格納、またはBLOBから階層リストを取り出すには、LIST TO BLOB と BLOB to list を使用します。

警告: 変数を格納するためにBLOBを使用したら、格納されたBLOBの内容を読み出すにはBLOB TO VARIABLE コマンドを使用しなければなりません。変数は4D内部形式を使用してBLOBに格納されるためです。

呼び出し後、変数が正常に格納された場合には、システム変数OKは1に設定されます。変数を格納するために必要なメモリがない等の理由で処理が実行できなかった場合には、システム変数OKは0に設定されます。

プラットフォーム独立互換性に関する注意

VARIABLE TO BLOB と BLOB TO VARIABLE は4Dの内部フォーマットを使用してBLOBに格納された変数を取り扱います。この利点として、これら二つのコマンドを使用する際、プラットフォーム間の バイトスワップを気にする必要はありません。言い換えればこれらのコマンドを使用してWindowsで作成されたBLOBを Mac OS で使用す る、あるいはその逆を行うことができます。

例題 1

以下の2つのプロジェクトメソッドを使用すると、ディスク上のドキュメントへすばやく配列を格納、またはドキュメントからすばやく配列を取得できます:

```
  ` SAVE ARRAY project method
  ` SAVE ARRAY ( String ; Pointer )
  ` SAVE ARRAY ( Document ; -> Array )
 C_STRING(255;$1)
 C_POINTER($2)
 C_BLOB($vxArrayData)
 VARIABLE TO BLOB($2->;$vxArrayData) ` Store the array into the BLOB
 COMPRESS BLOB($vxArrayData) ` Compress the BLOB
 BLOB TO DOCUMENT($1;$vxArrayData) ` Save the BLOB on disk

  ` LOAD ARRAY project method
  ` LOAD ARRAY ( String ; Pointer )
  ` LOAD ARRAY ( Document ; -> Array )
 C_STRING(255;$1)
 C_POINTER($2)
 C_BLOB($vxArrayData)
 DOCUMENT TO BLOB($1;$vxArrayData) ` Load the BLOB from the disk
 EXPAND BLOB($vxArrayData) ` Expand the BLOB
 BLOB TO VARIABLE($vxArrayData;$2->) ` Retrieve the array from the BLOB
```

上記のメソッドをアプリケーションに追加すれば、以下のように記述することができます:

```
 ARRAY STRING(...;asAnyArray;...)
  ` ...
 SAVE ARRAY($vsDocName;->asAnyArray)
  ` ...
 LOAD ARRAY($vsDocName;->asAnyArray)
```

例題 2

以下の2つのプロジェクトメソッドを使用すると、任意の変数（1～n個）をすばやく、BLOBへ格納／復元することができます:

```
  ` STORE VARIABLES INTO BLOB project method
  ` STORE VARIABLES INTO BLOB ( Pointer { ; Pointer ... { ; Pointer } } )
  ` STORE VARIABLES INTO BLOB ( BLOB { ; Var1 ... { ; Var2 } } )
 C_POINTER(${1})
 C_LONGINT($vlParam)

 SET BLOB SIZE($1->;0)
 For($vlParam;2;Count parameters)
    VARIABLE TO BLOB(${$vlParam}->;$1->;*)
 End for


  ` RETRIEVE VARIABLES FROM BLOB project method
  ` RETRIEVE VARIABLES FROM BLOB ( Pointer { ; Pointer ... { ; Pointer } } )
  ` RETRIEVE VARIABLES FROM BLOB ( BLOB { ; Var1 ... { ; Var2 } } )
 C_POINTER(${1})
 C_LONGINT($vlParam;$vlOffset)

 $vlOffset:=0
 For($vlParam;2;Count parameters)
    BLOB TO VARIABLE($1->;${$vlParam}->;$vlOffset)
 End for
```

これらのメソッドをアプリケーションに追加すれば、以下のように記述することができます:

```
 STORE VARIABLES INTO BLOB(->vxBLOB;->vgPicture;->asAnArray;->alAnotherArray)
  ` ...
 RETRIEVE VARIABLES FROM BLOB(->vxBLOB;->vgPicture;->asAnArray;->alAnotherArray)
```

システム変数およびセット

変数が正しく格納されるとOK変数は1に設定されます。そうでなければ0に設定されます。

HTTPクライアント

- HTTP AUTHENTICATE
- HTTP Get Updated 14.0
- HTTP Get certificates folder New 14.0
- HTTP GET OPTION
- HTTP Request Updated 14.0
- HTTP SET CERTIFICATES FOLDER New 14.0
- HTTP SET OPTION

HTTP AUTHENTICATE

HTTP AUTHENTICATE (name ; password {; authMethod} {; *})

引数	型		説明
name	テキスト	⇒	ユーザー名
password	テキスト	⇒	ユーザーパスワード
authMethod	倍長整数	⇒	認証方式: 0または省略時=指定しない、1=BASIC、2=DIGEST
*	演算子	⇒	指定時: プロクシ認証

説明

HTTP AUTHENTICATEコマンドはクライアントアプリケーションの認証を要求するサーバーへのHTTPリクエストを可能にします。

nameとpassword引数には必要な認証情報 (ユーザー名とパスワード) を渡します。この情報はエンコードされ、HTTP RequestまたはHTTP Getを使用して送信されるHTTPリクエストに追加されます。

オプションのauthMethod引数を使用して、使用する認証メソッドを指定できます。HTTP Clientテーマの以下の定数を使用できます:

定数	型	値	コメント
HTTP Basic	倍長整数	1	BASIC認証メソッドを使用する
HTTP Digest	倍長整数	2	DIGEST認証メソッドを使用する

authMethod引数を省略するか0を渡した場合、適切なメソッドを4Dが選択します。この場合4Dは認証メソッドをネゴシエートするために追加のリクエストを送信します。

* 引数を渡すと、認証情報はHTTPプロクシに提供されることを意味します。この設定はクライアントとHTTPサーバーの間に認証を必要とするプロクシが存在する場合に必要となります。サーバー自身も認証を行い場合、二重の認証が必要となります。

デフォルトで認証情報はプロセスが終了するまで保持されます。HTTP RequestやHTTP Getの呼び出し毎に**HTTP AUTHENTICATE**を使用する必要はありません。しかしHTTP SET OPTIONコマンドを使用すればこの情報を削除するよう指示することができます。この場合各リクエスト毎に**HTTP AUTHENTICATE**コマンドを呼び出す必要があります。

例題

認証付きリクエストの例:

```
  // DIGESTモード
 HTTP AUTHENTICATE("httpUser";"123";2)
  // プロクシ経由デフォルトモード
 HTTP AUTHENTICATE("ProxyUser";"456";*)
 $httpStatus:=HTTP Get(...)
```

HTTP Get

HTTP Get (url ; response {; headerNames ; headerValues}{; *}) -> 戻り値

引数	型		説明
url	テキスト	→	リクエスト送信先URL
response	テキスト, BLOB, ピクチャー, Object	←	リクエストの結果
headerNames	テキスト配列	→ ←	リクエストのヘッダー名 返されたヘッダー名
headerValues	テキスト配列	→ ←	リクエストのヘッダー値 返されたヘッダー値
*	演算子	→	指定時: 接続を保持する (keep-alive) 省略時: 自動で接続を閉じる
戻り値	倍長整数	↩	HTTPステータスコード

説明

HTTP Getコマンドは指定したURLにHTTP GETリクエストを送信し、HTTPサーバーからのレスポンスを処理します。

url 引数にはリクエストの送信先URLを渡します。シンタックスは以下の通りです:

```
http://[{user}:[{password}]@]host[:{port}][/{path}][?{queryString}]
```

例えば以下のような文字列を渡せます:

```
http://www.myserver.com
http://www.myserver.com/path
http://www.myserver.com/path?name=jones
https://www.myserver.com/login (*)
http://123.45.67.89:8083
http://john:smith@123.45.67.89:8083
```

(*) HTTPSリクエストを行う場合でも、証明書の発行局は検証されません。

コマンド実行後、response 引数はサーバーから返される結果を受け取ります。この結果はレスポンスからヘッダーを取り除いたボディー部です。
responseには異なる型の変数を渡すことができます:

- テキスト: 期待される結果がテキストの場合。
- BLOB: 期待される結果がバイナリの場合。
- ピクチャー: 期待される結果がピクチャーの場合。
- オブジェクト:期待される結果が C_OBJECT オブジェクトの場合。

BLOBを渡した場合、テキストやピクチャー、その他 (.wav, .zip, etc.) どのようなタイプのコンテンツでも受け取ることができます。受け取った内容を復元するのは開発者の仕事です (ヘッダーはBLOBに含まれません)。サーバーから返された結果がresponseの変数型に対応しない場合、変数は空のままです。C_OBJECT 型のオブジェクトを渡したとき、リクエストが結果を "application/json" (または "*何か*json") コンテン型とともに返してきた場合、4Dはオブジェクトを生成するためにJSONコンテンツを解析しようとします。

headerNames と headerValuesにはリクエストヘッダーの名前と値をそれぞれ格納した配列を渡します。
このコマンド実行後、これらの配列にはHTTPサーバーから返されたレスポンスのヘッダー情報で置き換えられます。これにより特にCookieを管理できます。

* 引数を使用してサーバー接続時にkeep-aliveメカニズムを有効にできます。デフォルトではこの引数が省略されると、keep-aliveは有効になりません。

コマンドからは標準のHTTPステータスコードが返されます (200=OK等)。HTTPステータスコードについてはRFC 2616を参照してください。
ネットワークに関連する理由 (DNS解決に失敗した、サーバーに接続できないなど...) により、サーバーに接続できない場合、コマンドは0を返し、エラーが生成されます。コマンドがサーバー証明書を使用していてその証明書が無効(失効しているか取り消されている)場合、コマンドは0を返しエラー901 "Server certificate invalid"が返されます。このエラーはON ERR CALLコマンドを使用してインストールされたエラー処理メソッドで処理できます。

例題 1

4D Webサイトから4Dロゴを取得する:

```
C_TEXT(URLPic_t)
URLPic_t:="http://www.4d.com/sites/all/themes/dimention/images/home/logo4D.jpg"
ARRAY TEXT(HeaderNames_at;0)
ARRAY TEXT(HeaderValues_at;0)
C_PICTURE(Pic_i)
$httpResponse:=HTTP Get(URLPic_t;Pic_i;HeaderNames_at;HeaderValues_at)
```

例題 2

RFCを取得する:

```
C_TEXT(URLText_t)
C_TEXT(Text_t)
URLText_t:="http://tools.ietf.org/rfc/rfc1.txt"
ARRAY TEXT(HeaderNames_at;0)
ARRAY TEXT(HeaderValues_at;0)
$httpResponse:=HTTP Get(URLText_t;Text_t;HeaderNames_at;HeaderValues_at)
```

例題 3

動画を取得する:

```
C_BLOB(vBlob)
$httpResponse:=HTTP Get("http://www.example.com/video.flv";vBlob)
BLOB TO DOCUMENT(vBlob;"video.flv")
```

HTTP Get certificates folder

HTTP Get certificates folder -> 戻り値

引数	型		説明
戻り値	テキスト	↩	アクティブな証明書フォルダの完全なパス名

説明

HTTP Get certificates folderコマンドは、アクティブなクライアント証明書フォルダの完全なパス名を返します。

デフォルトでは、4D はストラクチャーファイルのとなりに作成された "ClientCertificatesFolder" というフォルダを使用します(必要がある場合にのみ作成されます)。HTTP SET CERTIFICATES FOLDER コマンドを使用することによって、カレントプロセスにおいてユーザー定義のフォルダを作成することができます。

例題

証明書フォルダを一時的に変更したい場合を考えます:

```
C_TEXT($certifFolder)
$certifFolder :=HTTP Get certificates folder //save current folder
HTTP SET CERTIFICATES FOLDER("C:/temp/certifTempo/")
... // execution of specific requests
HTTP SET CERTIFICATES FOLDER($certifFolder) //restore previous folder
```

HTTP GET OPTION

HTTP GET OPTION (option ; value)

引数	型		説明
option	倍長整数	→	取得するオプションのコード
value	倍長整数	←	オプションの現在の値

説明

HTTP GET OPTIONコマンドは (HTTP GetやHTTP Requestコマンドで実行される次のリクエストでクライアントにより使用される) HTTPオプションの現在値を返します。現在値はデフォルト設定あるいはHTTP SET OPTIONコマンドで設定された値です。

注: 設定されるオプションはカレントプロセスで有効です。コンポーネントから呼び出された場合、コンポーネント内で有効となります。

option 引数には値を取得したいオプションの番号を渡します。HTTP Clientテーマの以下の定義済み定数を使用できます:

定数	型	値	コメント
HTTP Compression	倍長整数	6	値 = 0 (圧縮しない) または 1 (圧縮する)。デフォルト値: 0 このオプションを使用して、クライアント/サーバー間通信を効率化するための圧縮メカニズムの有効/無効を切り替えることができます。このメカニズムが有効になっていると、HTTPクライアントはサーバーのレスポンスに応じてdeflateまたはgzip圧縮を使用します。
HTTP Display auth dial	倍長整数	4	値 = 0 (ダイアログボックスを表示しない) または 1 (ダイアログボックスを表示する)。デフォルト値: 0 このオプションを使用して、HTTP GetやHTTP Requestを実行した際に認証ダイアログボックスを表示するかどうかを指定できます。デフォルトで認証ダイアログボックスは表示されず、HTTP AUTHENTICATEコマンドを使用して認証を行います。しかし認証ダイアログを表示してユーザーに認証IDを入力させたい場合、valueに1を渡します。ダイアログボックスは認証が必要な場合のみ表示されます。
HTTP Follow redirect	倍長整数	2	値 = 0 (リダイレクトを許可しない) または 1 (リダイレクトを許可する) デフォルト値 = 1
HTTP Max redirect	倍長整数	3	*値* = 受け入れるリダイレクト数の最大値 デフォルト値 = 2
HTTP Reset auth settings	倍長整数	5	値 = 0 (情報を削除しない) または 1 (情報を削除する)。デフォルト値: 0 このオプションを使用して4Dに、各HTTP GetやHTTP Requestコマンド実行毎に、ユーザーの認証情報 (ユーザー名、パスワード、認証メソッド) をリセットするよう指示できます。デフォルトでこれらの情報は保持され、各リクエストで再利用されます。value に1を渡すとコマンド実行毎にこれらの情報は削除されます。この設定に関わらず、これらの情報はプロセスが終了すると削除されます。
HTTP Timeout	倍長整数	1	値 = クライアントリクエストのタイムアウト (秒単位)。このタイムアウトはサーバーからのレスポンスを何秒待つかを指定します。タイムアウト時間を経過するとクライアントはセッションを閉じ、リクエストは失われます。 デフォルトでタイムアウトは20秒に設定されています。ネットワークの状態やリクエストの特性に応じてこの値を変更できます。

value 引数には現在の option 値を受け取る変数を渡します。

HTTP Request

HTTP Request (httpMethod ; url ; contents ; response {; headerNames ; headerValues}{; *}) -> 戻り値

引数	型		説明
httpMethod	テキスト	→	リクエストで使用するHTTPメソッド
url	テキスト	→	リクエストの送信先URL
contents	テキスト, BLOB, ピクチャー, Object	→	リクエストボディの内容
response	テキスト, BLOB, ピクチャー, Object	←	レスポンスの内容
headerNames	テキスト配列	→ ←	リクエストのヘッダー名 返されたヘッダー名
headerValues	テキスト配列	→ ←	リクエストのヘッダー値 返されたヘッダー値
*	演算子	→	指定時: 接続を保持する (keep-alive) 省略時: 自動で接続を閉じる
戻り値	倍長整数	↩	HTTPステータスコード

説明

HTTP Requestコマンドは指定したURLに任意のメソッドでHTTPリクエストを送信し、HTTPサーバーのレスポンスを処理することを可能にします。

httpMethod 引数にはHTTPリクエストのメソッドを渡します。HTTP Clientテーマの以下の定数を使用できます:

定数	型	値	コメント
HTTP DELETE Method	文字列	DELETE	RFC 2616参照
HTTP GET Method	文字列	GET	RFC 2616参照。HTTP Getを使用するのと同等。
HTTP HEAD Method	文字列	HEAD	RFC 2616参照
HTTP OPTIONS Method	文字列	OPTIONS	
HTTP POST Method	文字列	POST	RFC 2616参照
HTTP PUT Method	文字列	PUT	RFC 2616参照
HTTP TRACE Method	文字列	TRACE	RFC 2616参照

リクエストの送信先を url 引数に渡します。使用されるシンタックスは以下の通りです:

```
http://[{user}:[{password}]@]host[:{port}][/{path}][?{queryString}]
```

例えば以下の文字列を渡すことができます:

```
http://www.myserver.com
http://www.myserver.com/path
http://www.myserver.com/path?name=jones
https://www.myserver.com/login (*)
http://123.45.67.89:8083
http://john:smith@123.45.67.89:8083
```

(*) HTTPSリクエストを行う場合でも、証明書の発行局は検証されません。

contents 引数にはリクエストのボディを渡します。この引数に渡されるデータはリクエストのHTTPメソッドにより異なります。
テキストやBLOB、ピクチャー、オブジェクトデータを送信できます。*content-type* が指定されない場合、以下のタイプが使用されます:

- テキスト: text/plain - UTF8
- BLOB: application/byte-stream
- ピクチャー: 既知のMIMEタイプ (best for Web).
- C_OBJECT オブジェクト: application/json

コマンド実行後、response 引数はサーバーから返された結果を受け取ります。この結果はレスポンスからヘッダーを取り除いたボディー部になります。
responseには異なる型のレスポンスを渡せます:

- テキスト: 期待される結果がテキストまたはXMLツリー参照の場合。
- BLOB: 期待される結果がバイナリの場合。
- ピクチャー: 期待される結果がピクチャーの場合。
- C_OBJECT オブジェクト: 期待される結果がオブジェクトの場合。

C_OBJECT 型の変数を response 引数に渡したとき、リクエストが "application/json" (または "something/json") コンテンツ型と結果を返してきた場合、4D はオブジェクトを生成するためにJSON コンテンツを解析しようとします。

サーバーから返された結果がresponseの変数型に対応しない場合、変数は空のままでOKシステム変数に0が設定されます。

headerNames と headerValuesにはリクエストヘッダーの名前と値をそれぞれ格納した配列を渡します。
このコマンド実行後、これらの配列にはHTTPサーバーから返されたレスポンスのヘッダー情報で置き換えられます。これにより特にCookieを管理できます。

* 引数を使用してサーバー接続時にkeep-aliveメカニズムを有効にできます。デフォルトではこの引数が省略されると、keep-aliveは有効になりません。

コマンドからは標準のHTTPステータスコードが返されます (200=OK等)。HTTPステータスコードについてはRFC 2616を参照してください。
ネットワークに関連する理由 (DNS解決に失敗した、サーバーに接続できないなど...) により、サーバーに接続できない場合、コマンドは0を返し、エラーが生成されます。このエラーはON ERR CALLコマンドを使用してインストールされたエラー処理メソッドで処理できます。コマンドがサーバー証明書を使用していてその証明書が有効でない(失効または無効の)とき、コマンドは 0 を返しエラー 901 "Server certificate invalid" が返されます。これらのエラーはON ERR CALLコマンドを使用してインストールされたエラー処理メソッドで処理できます。

例題 1

リモートデータベースからレコード削除を要求する:

```
$body_t:="{record_id:25}"
$httpStatus_l:=HTTP Request(HTTP DELETE Method;"database.example.com";$body_t)
```

注: サーバー側では受け付けたリクエストに応じて適切な処理を実装しなければなりません。**HTTP Request**はリクエストの送信と返される結果を処理するだけです。

例題 2

リモートデータベースにレコード追加を要求する:

```
$body_t:="{fName:'john',fName:'Doe'}"
$httpStatus_l:=HTTP Request(HTTP PUT Method;"database.example.com";$body_t)
```

注: サーバー側では受け付けたリクエストに応じて適切な処理を実装しなければなりません。**HTTP Request**はリクエストの送信と返される結果を処理するだけです。

例題 3

リモートデータベースにJSON形式のレコードを追加するリクエスト:

```
C_OBJECT($content)
OB SET($content;"lastname";"Doe";"firstname";"John")
$result:=HTTP Request(HTTP PUT Method;"database.example.com";$content;$response)
```

HTTP SET CERTIFICATES FOLDER

HTTP SET CERTIFICATES FOLDER (certificatesFolder)

引数	型		説明
certificatesFolder	テキスト	→	クライアント証明書フォルダのパス名と名前

説明

HTTP SET CERTIFICATES FOLDER コマンドは、カレントのプロセスにおいてアクティブな証明書フォルダを変更します。

クライアント証明書フォルダはWebサーバーからクライアント証明書の要求があったときに4Dが探しに行くフォルダです。デフォルトでは、**HTTP SET CERTIFICATES FOLDER** コマンドが実行されていない限り、4Dはストラクチャーファイルの隣に作成される "ClientCertificatesFolder" というフォルダを使用します。このフォルダは必要がある場合にのみ作成されます。

4D v14では、複数のクライアント証明書を使用できるようになりました。

certificatesFolder 引数には、クライアント証明書を内包しているユーザー定義のフォルダのパス名を渡します。アプリケーションのストラクチャーファイルからの相対パスか、または絶対パスを渡すことができます。パスは以下の例のようにシステムシンタックスに従って書かれている必要があります:

- (OS X): Disk:Applications:myserv:folder
- (Windows): C:\Applications\myserv\folder

指定されたフォルダが定義された場所にない場合は、自動的に作成されます。

このコマンドが実行されると新しいパスは直ちに有効になります(アプリケーションを再起動する必要はありません)。このコマンドのスコープはカレントプロセスとなっているので、呼び出されたときに必要な数だけSOAPまたはHTTPリクエストを実行することができます。異なるサーバーで複数のリクエストを同時に実行したい場合は、複数のプロセスを作成して下さい。

デフォルトの証明書フォルダを使用したい場合には、certificatesFolder に空の文字列を渡して下さい。

certificatesFolder に渡したパス名が有効でない場合、エラーが生成されます。このエラーは ON ERR CALL によって実装されたエラーハンドリングメソッドによって割り込むことができます。

HTTP SET OPTION

HTTP SET OPTION (option ; value)

引数	型		説明
option	倍長整数	⇒	設定するオプションのコード
value	倍長整数	⇒	オプションの値

説明

HTTP SET OPTIONコマンドは (HTTP GetやHTTP Requestコマンドで実行される次のリクエストでクライアントにより使用される) HTTPオプションの値を設定します。設定するオプションの数だけこのコマンドを呼び出します。

注: 設定されるオプションはカレントプロセスに対し有効です。コンポーネントの場合そのコンポーネント内で有効です。

option 引数には設定するオプションの番号を、value 引数にはオプションの新しい値を渡します。option 引数にはHTTP Clientテーマの以下の定義済み変数のいずれかを指定できます:

定数	型	値	コメント
HTTP Compression	倍長整数	6	値 = 0 (圧縮しない) または 1 (圧縮する)。デフォルト値: 0 このオプションを使用して、クライアント/サーバー間通信を効率化するための圧縮メカニズムの有効/無効を切り替えることができます。このメカニズムが有効になっていると、HTTPクライアントはサーバーのレスポンスに応じてdeflateまたはgzip圧縮を使用します。
HTTP Display auth dial	倍長整数	4	値 = 0 (ダイアログボックスを表示しない) または 1 (ダイアログボックスを表示する)。デフォルト値: 0 このオプションを使用して、HTTP GetやHTTP Requestを実行した際に認証ダイアログボックスを表示するかどうかを指定できます。デフォルトで認証ダイアログボックスは表示されず、HTTP AUTHENTICATEコマンドを使用して認証を行います。しかし認証ダイアログを表示してユーザーに認証IDを入力させたい場合、valueに1を渡します。ダイアログボックスは認証が必要な場合のみ表示されます。
HTTP Follow redirect	倍長整数	2	値 = 0 (リダイレクトを許可しない) または 1 (リダイレクトを許可する) デフォルト値 = 1
HTTP Max redirect	倍長整数	3	*値* = 受け入れるリダイレクト数の最大値 デフォルト値 = 2
HTTP Reset auth settings	倍長整数	5	値 = 0 (情報を削除しない) または 1 (情報を削除する)。デフォルト値: 0 このオプションを使用して4Dに、各HTTP GetやHTTP Requestコマンド実行毎に、ユーザーの認証情報 (ユーザー名、パスワード、認証メソッド) をリセットするよう指示できます。デフォルトでこれらの情報は保持され、各リクエストで再利用されます。value に1を渡すとコマンド実行毎にこれらの情報は削除されます。この設定に関わらず、これらの情報はプロセスが終了すると削除されます。
HTTP Timeout	倍長整数	1	値 = クライアントリクエストのタイムアウト (秒単位)。このタイムアウトはサーバーからのレスポンスを何秒待つかを指定します。タイムアウト時間を経過するとクライアントはセッションを閉じ、リクエストは失われます。 デフォルトでタイムアウトは20秒に設定されています。ネットワークの状態やリクエストの特性に応じてこの値を変更できます。

オプションはどのような順番で設定してもかまいません。おなじオプションを複数回設定した場合、最後に設定された値が使用されます。

JSON

- JSON コマンドについての概要
- JSON Parse New 14.0
- JSON PARSE ARRAY New 14.0
- JSON Stringify New 14.0
- JSON Stringify array New 14.0
- JSON TO SELECTION New 14.0
- Selection to JSON New 14.0

JSON コマンドについての概要

JSON コマンドは、JSONフォーマットのランゲージオブジェクトを生成・解析するためのものです。言い換えると、このフォーマットを使用することによって4Dデータベース(データとストラクチャ)にWebブラウザを使ってアクセスすることが出来るようになります。

構造化されたオブジェクトの使えるようになったことは4D v14 のランゲージの大きな新機能の一つであり、構造化されたデータの受け渡しを容易にするのが目的です。"JSON"テーマ内のコマンドを使用することによって4DでJSONオブジェクトを直接使うことが出来るようになりました。それでいながら、"native"オブジェクト(JSONにインスパイヤされた構造をもつオブジェクト)を扱うこともで、これにより全てのランゲージとデータの受け渡しができるようになっています。詳細は オブジェクト(ランゲージ) の章を参照して下さい。

JSONについての概要

"JSON (JavaScript Object Notation)とは、ECMAScript 表記法に由来する、標準的なテキストベースのデータ表記法である。" (引用:Wikipedia英語版)
JSON は特定のプログラミング言語に依存しませんが、それでいてC++ やJavaScript、Perl等といった言語の使用者にはなじみ深い慣習を使用しています。データの受け渡し特に特化したフォーマットであると言えるでしょう。

この章ではJSONの表記法の原理についての概略を記載しています。JSONの表記法についての完全な情報については、こちらのWebサイトを参照して下さい。: www.json.org/index.html

JSON 記法

JSON の記法は以下の原則に従うようにできています。:

- データは、名前と値がペアになるようにできています。
- データはカンマ(,)で区切られています。
- オブジェクトは中カッコ({})によって定義されます。
- 配列は大カッコ([ ])によって定義されます。

JSON プロパティ

JSON データは、名前/値(もしくはキー/値)のペアという形で表現されます。名前/値のペアの中は、二重引用符("")で挟まれたフィールド名のあと、コロン(:)、そして同じく二重引用符("")で挟まれた値が続きます。以下の様な形です:

```
"firstName":"John"
```

これはJavaScriptでの以下の表記に相当しています:

```
firstName="John"
```

プロパティ名では大文字と小文字は区別されることに注意してください。"FirstName" の代わりに "firstName" と表記すると、これは別のデータを意味することになります。

JSON データ型

JSON では以下の型の値に対応しています:

型	説明	詳細
文字列	二重引用符(")とバックスラッシュ(/)以外ならどんなUnicode文字も使用できます。 プロパティ名などの値は二重引用符(")に挟まれる決まりになっています。例えば、"city":"Paris"	制御文字には \ が使用されます: \" = quotes \\ = backslash \/ = slash \b = backspace \f = formfeed \n = line break \r = carriage return \t = tab \u = four hexadecimal digits
数値	整数または浮動少数	数字に関してはCやJavaに近いですが、8進法数と16進法数は使用できません。
オブジェクト	{ }	
配列	[ ]	
ブール	true または false	
null	null	

JSON オブジェクト

JSON オブジェクトは中カッコ({ })によって定義され、必要な数だけ名前/値のペアを内包することが出来ます。表記は以下のようになります:

```
{ "firstName":"John" , "lastName":"Doe" }
```

JSON 配列

JSON 配列は大カッコ([ ])によって定義されます。それぞれの配列は、未定数のオブジェクトを内包することができます:

```
{
    "employees": [
        { "firstName":"John" , "lastName":"Doe" },
        { "firstName":"Anna" , "lastName":"Smith" },
        { "firstName":"Peter" , "lastName":"Jones" }
    ]
}
```

タイムゾーンのサポート

4D の日付とJSON感でデータの変換が行われる際、デフォルトで変換を行ったマシンのタイムゾーンに沿って(JavaScriptに従って)変換されます。例えば、フランス(GMT+2)では !23/08/2013! を変換すると"2013-08-22T22:00:00Z" という結果が得られます(逆もまた然りです)。

例えばSET DATABASE PARAMETER コマンドなどを使用してこの機能を変更し、書き出し処理を実行する際にタイムゾーンを考慮しないようにすることもできます。

4D/JSON 間の日付の変換についてのより詳細な情報は、JavaScript の日付の変換 を参照して下さい。

JSON Parse

JSON Parse (jsonString {; type}) -> 戻り値

引数	型		説明
jsonString	文字	→	解析したいJSON文字列
type	倍長整数	→	値を変換したい型
戻り値	Object, テキスト, ブール, ポインター, 実数	↩	JSON文字列から取り出した値

説明

JSON Parseコマンドは、JSONフォーマットの文字列の中身を解析し、4Dのフィールドまたは変数へと保存可能な値を取り出します。このコマンドはJSONデータをデシリアライズします。つまりJSON Stringifyコマンドと逆の挙動をします。

jsonString 引数には、解析をしたいJSONフォーマットの文字列を渡します。この文字列が正しくフォーマットされていないと、解析はエラーを生成します。

注: ポインターを使用した場合、**JSON Parse**を使用する前に JSON Stringify コマンドを使用する必要があります。

type 引数を省略した場合、結果を保存するのに使用する変数またはフィールドが定義されていれば、4Dは自動的に、取得した値をその型へと変換します。そうでない場合は4Dは型を推測します。type 引数を渡す事によって変換する型を強制的に指定することもできます。渡す場合は、Field and Variable Typesテーマ内にある以下の定数のどれか一つを渡して下さい。

定数	型	値
Is Boolean	倍長整数	6
Is Date	倍長整数	4
Is LongInt	倍長整数	9
Is Real	倍長整数	1
Is Text	倍長整数	2

注:

- 実数型の値は ±10.421e±10の範囲内に収まっていなければなりません。
- テキスト型の値の場合、全ての特殊文字は引用符を含めエスケープされてなければなりません(例を参照して下さい)。
- JSON の日付は全て "\"YYYY-MM-DDTHH:mm:ssZ\""のフォーマットでなければなりません。このコマンドは4Dの日付がGMTではなくローカル時刻を含むことを考慮します。

例題 1

単純な変換の例:

```
C_REAL($r)
$r:=JSON Parse("42.17") //$r = 42,17 (Real)

C_LONGINT($el)
$el:=JSON Parse("120.13";Is LongInt) //$el=120

C_TEXT($t)
$t:=JSON Parse("\"Year 42\"";Is Text) // $t="Year 42" (text)

C_OBJECT($o)
$o:=JSON Parse("{\"name\":\"jean\"}")
 // $o = {"name":"john"} (4D object)

C_BOOLEAN($b)
$b:=JSON Parse("{\"manager\":true}";Is Boolean) // $b=true
```

例題 2

日付型のデータの変換の例:

```
$test:=JSON Parse("\"1990-12-25T12:00:00Z\"")
 // $test=1990-12-25T12:00:00Z
C_DATE($date)
$date:=JSON Parse("\"2008-01-01T12:00:00Z\"";Is Date)
 //$date=01/01/08
```

例題 3

ここでは JSON Stringify と **JSON Parse** コマンドの併用例を紹介しています:

```
C_TEXT($MyContact)
C_OBJECT($Contact)

 // JSON Stringify: JSON オブジェクトからJSON文字列への変換
$MyContact:=JSON Stringify("{\"name\":\"Monroe\",\"firstname\":\"Alan\"}")
 // $MyContact = "{\\"name\\":\\"Monroe\\",\\"firstname\\":\\"Alan\\"}"
 // JSON Parse: JSON文字列からJSONオブジェクトへの変換
$Contact:=JSON Parse("{\"name\":\"Monroe\",\"firstname\":\"Alan\"}")
 // $Contact = {"name":"Monroe","firstname":"Alan"}
```

JSON PARSE ARRAY

JSON PARSE ARRAY (jsonString ; objArray)

引数	型		説明
jsonString	文字	→	解析したいJSON文字列
objArray	Object array	←	JSON文字列を解析した結果を含む配列

説明

JSON PARSE ARRAYコマンドは、JSONフォーマットの文字列の中身を解析し、取り出したデータを objArray の配列の中へと入れます。このコマンドはJSONデータをデシリアライズします。JSON Stringify array配列コマンドと逆の挙動をします。

jsonString 引数には、解析したいJSONフォーマットの文字列を渡します。この文字列は正しいフォーマットで書かれてる必要があり、そうでない場合にはエラーが生成されます。

objArray 引数には、解析結果を受け取るオブジェクトを渡します。

例題

この例では、テーブル内のレコードのフィールドからのデータが取り出され、オブジェクト配列に保存されます。

```
C_OBJECT($ref)
ARRAY OBJECT($sel;0)
ARRAY OBJECT($sel2;0)
C_TEXT(v_String)

OB SET($ref;"name";->[Company]Company Name)
OB SET($ref;"city";->[Company]City)

While(Not(End selection([Company])))
   $ref_company:=OB Copy($ref;True)
   APPEND TO ARRAY($sel;$ref_company)
 // $sel{1}={"name":"4D SAS","city":"Clichy"}
 // $sel{2}={"name":"MyComp","city":"Lyon"}
 // ...
   NEXT RECORD([Company])
End while

v_String:=JSON Stringify array($sel)
 // v_String= [{"name":"4D SAS","city":"Clichy"},{"name":"MyComp","city":"Lyon"}...]
JSON PARSE ARRAY(v_String;$sel2)
 // $sel2{1}={"name":"4D SAS","city":"Clichy"}
 // $sel2{2}={"name":"MyComp","city":"Lyon"}
 //...
```

JSON Stringify

JSON Stringify (value {; *}) -> 戻り値

引数	型		説明
value	Object, Object array, 文字, 実数, 日付, 時間	⇒	JSON文字列に変換したいデータ
*	演算子	⇒	整形フォーマット
戻り値	テキスト	↩	シリアライズされたJSONテキストを含む文字列

説明

JSON Stringifyコマンドは、value 引数を、JSON文字列へと変換します。このコマンドはデータをJSONへとシリアライズします。つまり JSON Parse コマンドと逆の挙動をします。

シリアライズしたいデータをvalue に渡します。スカラー値(文字列、数字、日付または時間)または4Dオブジェクト(またはオブジェクト配列)を渡す事ができます。

オブジェクトを渡す場合、全ての型の値を含めることができます(JSON データ型 の賞を参照して下さい)。JSONフォーマットは以下の規則に則っている必要があります。

- 文字列の値は引用符で囲われている必要があります。ユニコード文字は全て使用できますが、特殊文字はバックスラッシュでエスケープする必要があります。
- 数字は ±10.421e±10の範囲内におさまっていなければなりません。
- ブール型:"true"または"false"の文字列
- フィールド、変数、配列へのポインター(ポインターは文字列化したときに評価されます)。
- 日付:テキスト型
- 時間:実数型

任意の* 演算子を渡すことによって、戻り値の文字列にフォーマット文字を含めることができます。これによりJSONデータの表示が改善されます(いわゆる整形フォーマティングです)。

例題 1

スカラー値の変換:

```
$vc:=JSON Stringify("Eureka!") // "Eureka!"
$vel:=JSON Stringify(120) // "120"
$vd:=JSON Stringify(!28/08/2013!) //  "2013-08-27T22:00:00Z"
$vh:=JSON Stringify(?20:00:00?) // "72000000" seconds since midnight
```

例題 2

特殊文字を含む文字列の変換:

```
$s:=JSON Stringify("{\"name\":\"john\"}")
 // $s="{\\"name\\":\\"john\\"}"
$p:=JSON Parse($s)
 // $p={"name":"john"}
```

例題 3

変数へのポインターの使用例:

```
C_OBJECT($MyTestVar)
C_TEXT($name ;$jsonstring )
OB SET($MyTestVar;"name";->$name) // object definition
 // $MyTestVar= {"name":"->$name"}

$jsonstring :=JSON Stringify($MyTestVar)
 // $jsonstring ="{"name":""}"
 //...

$name:="Smith"
$jsonstring :=JSON Stringify($MyTestVar)
 //$jsonstring = "{"name" : "Smith"}"
```

例題 4

4Dオブジェクトのシリアライズ:

```
C_TEXT($varjsonTextserialized)
C_OBJECT($Contact)
OB SET($Contact;"firstname";"Alan")
OB SET($Contact;"lastname";"Monroe")
OB SET($Contact;"age";40)
OB SET($Contact;"phone";"[555-0100,555-0120]")

$varjsonTextserialized:=JSON Stringify($Contact)

 // $varjsonTextserialized = "{"lastname":"Monroe","phone":"[555-0100,
 // 555-0120]","age":40,"firstname":"Alan"}"
```

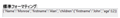
例題 5

4Dオブジェクトを、* 演算子ありとなしでシリアライズした場合の例:

```
C_TEXT($MyContact)
C_TEXT($MyPContact)
C_OBJECT($Contact;$Children)
OB SET($Contact;"lastname";"Monroe";"firstname";"Alan")
OB SET($Children;"firstname";"Jim";"age";"12")
OB SET($Contact;"children";$Children)
$MyContact:=JSON Stringify($Contact)
$MyPContact:=JSON Stringify($Contact;*)
 //$MyContact= {"lastname":"Monroe","firstname":"Alan","children":{"firstname":"John","age":"12"}}
 //$MyPContact= {\n\t"lastname": "Monroe",\n\t"firstname": "Alan",\n\t"children": {\n\t\t"firstname":
"John",\n\t\t"age": "12"\n\t}\n}
```

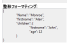
このフォーマティングを使った場合の利点はJSONがWebエリアに表示されれば一目瞭然です。

- 標準フォーマッティング:



```
{"Name":"Monroe","firstname":"Alan","children":{"firstname":"John","age":12}}
```

- 整形フォーマッティング:



```
{
		"Name": "Monroe",
		"firstname": "Alan",
		"children": {
				"firstname": "John",
				"age":12
		}
}
```

JSON Stringify array

JSON Stringify array (array {; *}) -> 戻り値

引数	型		説明
array	テキスト配列, 実数配列, ブール配列, ポインター配列, Object array	→	内容をシリアライズしたい配列
*	演算子	→	整形フォーマット
戻り値	テキスト	↩	シリアライズされたJSON配列を含む文字列

説明

JSON Stringify arrayコマンドは、4D配列 array をシリアライズされたJSON配列へと変換します。このコマンドは JSON PARSE ARRAYコマンドと逆の挙動をします。

array 引数にはシリアライズしたいデータを含む4D配列を渡して下さい。この配列はテキスト、実数、ブール、ポインターまたはオブジェクト型である必要があります。

任意の * 引数を渡す事によって戻り値の文字列に整形フォーマッティングを使用することもできます。これはフォーマッティング文字を含めることよってWebページで表示される際のJSONの表示を改善するものです。

例題 1

テキスト配列の変換:

```
C_TEXT($jsonString)
ARRAY TEXT($ArrayFirstname;2)
$ArrayFirstname{1}:="John"
$ArrayFirstname{2}:="Jim"
$jsonString :=JSON Stringify array($ArrayFirstname)

 // $jsonString = "["John","Jim"]"
```

例題 2

数字を含むテキスト配列の変換:

```
ARRAY TEXT($phoneNumbers;0)
APPEND TO ARRAY($phoneNumbers ;"555-0100")
APPEND TO ARRAY($phoneNumbers ;"555-0120")
$string :=JSON Stringify array($phoneNumbers)
 // $string = "["555-0100","555-0120"]"
```

例題 3

オブジェクト配列の変換:

```
C_OBJECT($ref_john)
C_OBJECT($ref_jim)
ARRAY OBJECT($myArray;0)
OB SET($ref_john;"name";"John";"age";35)
OB SET($ref_jim;"name";"Jim";"age";40)
APPEND TO ARRAY($myArray ;$ref_john)
APPEND TO ARRAY($myArray ;$ref_jim)
$JsonString :=JSON Stringify array($myArray)
 // $JsonString = "[{"name":"John","age":35},{"name":"Jim","age":40}]"

 // Webページでの結果を見たい場合は
 // 任意の * 演算子を渡します:
$JsonStringPretty :=JSON Stringify array($myArray;*)
```

```
[
  {
    "name": "John",
    "age": 35
  },
  {
    "name": "Jim",
    "age": 40
  }
]
```

例題 4

オブジェクト配列内の4Dセレクションの変換:

```
C_OBJECT($jsonObject)
C_TEXT($jsonString)

QUERY([Company];[Company]Company Name="a@")
OB SET($jsonObject;"company name";->[Company]Company Name)
OB SET($jsonObject;"city";->[Company]City)
OB SET($jsonObject;"date";[Company]Date_input)
OB SET($jsonObject;"time";[Company]Time_input)
ARRAY OBJECT($arraySel;0)

While(Not(End selection([Company])))
   $ref_value:=OB Copy($jsonObject;True)
 // これらをコピーしない場合、値は空の文字列となります。
   APPEND TO ARRAY($arraySel;$ref_value)
 // それぞれの要素は選択した値を含みます。例えば、
 // $arraySel{1} = // {"company name":"APPLE","time":43200000,"city":
 // "Paris","date":"2012-08-02T00:00:00Z"}
   NEXT RECORD([Company])
End while

$jsonString:=JSON Stringify array($arraySel)
 // $jsonString = "[{"company name":"APPLE","time":43200000,"city":
 //"Paris","date":"2012-08-02T00:00:00Z"},{"company name":
 //"ALMANZA",...}]"
```

JSON TO SELECTION

JSON TO SELECTION (aTable ; jsonObject)

引数	型		説明
aTable	ポインター	→	4Dテーブルへのポインター
jsonObject	テキスト	→	JSON形式の文字列

説明

JSON TO SELECTIONコマンドは、JSONオブジェクトの中身を、aTable のレコードのセレクションへとコピーします。.

aTable を呼び出したときに指定したセレクションが存在していた場合、JSON オブジェクトの要素はオブジェクトの順番とレコードの順番に応じてコピーされます。JSON オブジェクトによって定義された要素の数がカレントセレクション内に存在するレコードより多い場合、新たにレコードが追加されます。レコードは、存在していた場合でも新規に追加した場合でも、自動的に保存されます。

警告: JSON TO SELECTION コマンドは既存のレコード内の情報を上書きしてしまうため、使用の際には注意が必要です。

コマンド実行中、保存先のレコードが他の処理などでロックされていた場合、その中身は変更されません。ロックされたレコードは、LockedSet システムセット内に保存されています。**JSON TO SELECTION**コマンドを実行し終えたあと、*LockedSet* 内にロックされたレコードが保存されているかどうか検証することが出来ます。

例題

JSON TO SELECTION コマンドを使用して、[Company] テーブルへレコードを追加します。

```
C_OBJECT($Object1;$Object2;$Object3;$Object4)
C_TEXT($ObjectString)
ARRAY OBJECT($arrayObject;0)

OB SET($Object1;"ID";"200";"Company Name";"4D SAS";"City";"Clichy")
APPEND TO ARRAY($arrayObject;$Object1)

OB SET($Object2;"ID";"201";"Company Name";"APPLE";"City";"Paris")
APPEND TO ARRAY($arrayObject;$Object2)

OB SET($Object3;"ID";"202";"Company Name";"IBM";"City";"London")
APPEND TO ARRAY($arrayObject;$Object3)

OB SET($Object4;"ID";"203";"Company Name";"MICROSOFT";"City";"New York")
APPEND TO ARRAY($arrayObject;$Object4)

$ObjectString:=JSON Stringify array($arrayObject)

 // $ObjectString = "[{"ID":"200","City":"Clichy","Company Name":"4D
 // SAS"},{"ID":"201","City":"Paris","Company Name":"APPLE"},{"ID":"202",
 //"City":"London","Company Name":"IBM"},{"ID":"203","City":"New
 //York","Company Name":"MICROSOFT"}]"

JSON TO SELECTION([Company];$ObjectString)
 // [Company] テーブル内に、ID、会社名、都市のフィールドを
 //持つレコードを4つ作成しました。
```

Selection to JSON

Selection to JSON (aTable {; aField}{; aField2 ; ... ; aFieldN}{; template}) -> 戻り値

引数	型		説明
aTable	テーブル	→	シリアライズするテーブル
aField	フィールド	→	シリアライズするフィールドを指定
template	Object	→	プロパティ名と取り出したいフィールドを指定するポインターをオブジェクトで渡す
戻り値	テキスト	←	シリアライズされた JSON 配列を含む文字列

説明

Selection to JSONコマンドは、aTable のカレントセレクションのためのフィールドの値を含んだJSON文字列を返します。

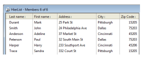
aTable 引数のみを渡した場合、テーブル内の、全てのフィールドの値のうち、JSON で表現できるものを JSON 文字列で返します。BLOB フィールドとピクチャーフィールドは無視されます。

aTable 内の一部のフィールドのみ取り出したい場合、aField 引数か、template 引数を使ってその部分を指定することが出来ます:

- aField: 一つ以上のフィールドをこの引数で指定して下さい。ここで指定したフィールドの値のみ JSON 文字列で返されます。
- template: 一つ以上の名前/値のペアを含んだ 4D オブジェクトを渡して下さい。この値に、取り出したいフィールドを指定するポインターを入れて指定します。(例3を参照して下さい。)

例題

以下のセレクションを JSON 文字列で表現する場合:



HierList - Members: 6 of 6

Last name :	First name :	Address :	City :	Zip Code :
Durant	Mark	25 Park St	Pittsburgh	15205
Smith	John	24 Philadelphia Ave	Dallas	75203
Anderson	Adeline	37 Market St	Cincinnati	45205
Peterson	Paul	32 South Main St	Dallas	75203
Harper	Harry	233 Southport Ave	Cincinnati	45206
Trace	Sandra	332 Court St	Pittsburgh	15205

1) [Members] テーブル内の全てのフィールドの値を取り出す場合:

```
$jsonString :=Selection to JSON([Members])
 // $jsonString =[{"LastName":"Durant","FirstName":"Mark","Address":
 //"25 Park St","Zip code":"15205","City":"Pittsburgh"},{"LastName":
 //"Smith","FirstName":"John","Address":"24 Philadelphia Ave","Zip code":
 //"75203","City":"Dallas"},{"LastName":"Anderson","FirstName"
 //:"Adeline","Address":"37 Market St","Zip code":"45205","City":"Cincinnati"},...]
```

2) フィールドを指定して、取り出すフィールドを二つだけに限定したい場合:

```
QUERY([Members];[Members]LastName="A@")
$jsonString :=Selection to JSON([Members];[Members]LastName;[Members]City)
 // $jsonString = [{"LastName":"Anderson","City":"Cincinnati"},{"LastName":"Albert","City":"Houston"}]
```

3) template 表記を用いて、一つのフィールドだけを取り出したい場合:

```
C_OBJECT($template)
OB SET($template;"LastName";->[Members]LastName) // 単一のフィールドを指定
ALL RECORDS([Members])
$jsonString :=Selection to JSON([Members];$template)
 // $jsonString =
[{"LastName":"Durant"},{"LastName":"Smith"},{"LastName":"Anderson"},{"LastName":"Albert"},{"LastName":"Leonard"},{"LastName":"Pradel"}]
```

PHP

- 4DでPHPスクリプトを実行する
- PHP Execute
- PHP GET FULL RESPONSE
- PHP GET OPTION
- PHP SET OPTION
- PHPモジュールのサポート

4DでPHPスクリプトを実行する

4DでPHPスクリプトを直接実行できます。これによりPHPで利用できる豊富なユーティリティライブラリを使用できるようになります。特に以下のような機能がライブラリに含まれます:

- 暗号化 (MD5) およびハッシュ
- ZIPファイルの処理
- ピクチャーの処理
- LDAP アクセス,
- COM アクセス(MS Officeドキュメントのコントロール), 等

このリストに挙げるもの以外にもあります。デフォルトで4Dで利用可能なPHPモジュールに関する完全なリストはPHPモジュールのサポート を参照してください。また追加のカスタムモジュールのインストールも可能です。

PHP スクリプトや関数を実行するにはPHP Executeコマンドを使用します。4D v12はデフォルトでバージョン 5.4.11 のPHPを含んでいます。実行されるスクリプトはこのバージョンおよびインストールされたモジュールと互換性が なければなりません。

PHPコマンドに関する完全な説明とシンタックスについては、インターネット上のPHPに関するドキュメントを参照してください。以下はリファレンスを参照できるサイトの例です:
http://us.php.net/manual/en/
http://phpdeveloper.org/
http://php.start4all.com/
http://php.resourceindex.com/Documentation/

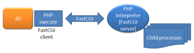
アーキテクチャ

4Dは、アプリケーションとPHPインタプリタ間のクライアント-サーバタイプのコミュニケーションプロトコルである、FastCGIでコンパイルされたPHPインタプリタを提供します。

PHPインタプリタは"子プロセス"と呼ばれる一連のシステム実行プロセスをコントロールします。これらのプロセスは4Dから送信されたリクエストを処理するために使用されます。リクエスト実行は同期的に行われます。最適化のため、デフォルトでは最大5つまでの子プロセスを同時に実行できます。この数値はデータベース設定やSET DATABASE PARAMETERコマンドで変更できます。Mac OSでは、これらのプロセスは最初のリクエストで起動され、PHPインタプリタにより永続的に保持されます。Windowsでは、必要に応じて4Dがプロセスを作成し、必要な時にリサイクルします。4Dはデフォルトで、起動と終了のPHPインタプリタのプロセスの処理を自動的にサポートします。

Note:PHP子プロセスが動作中である間に4Dプログラムが予期せず終了した場合、システムプロセス管理ウィンドウから手動でそれらを削除しなければなりません。

下図は4D v12の4D/PHPアーキテクチャを示しています:



このアーキテクチャは、4Dから定義済みのTCPアドレス (IPアドレスとポート番号) に送信される内部リクエストのシステムで動作します。同じマシン上で複数のPHPインタプリタが動作しているなどの場合は必要に応じてデータベース設定またはSET DATABASE PARAMETERコマンドを使用してこのアドレスを変更できます。

警告: 同じマシン上で2つの4Dアプリケーションを起動し、それぞれから (PHP Execute コマンドを使用して) PHPステートメントを実行する場合、FastCGI PHPインタープリタの待ち受けポートを変更して、それぞれのアプリケーションが異なるインタープリタを使用できるようにしなければなりません。そうでなければ、PHPステートメントを正しく実行することができず、4Dアプリケーションがフリーズすることもあります。

異なるPHPインタープリターとphp.iniを使用する

4Dが提供するものとは異なるPHPインタプリタを使用することができます。カスタムPHPインタプリタは2つの条件を満たさなければなりません:

- FastCGIでコンパイルされていなければなりません。
- 4Dと同じマシン上になければなりません。

カスタムPHPインタプリタを使用するには、指定されたアドレスとTCPポートで待ち受けを行うよう、また4Dの内部インタプリタを有効にしないよう設定します。これらのパラメタはデータベース設定 あるいはセッションごとにSET DATABASE PARAMETERコマンドで設定できます。もちろんこの場合、インタプリタの起動や動作は開発者が管理しなければなりません。

追加のモジュールを使用するなどのため、カスタマイズした*php.ini*設定ファイルを使用することもできます。php.iniファイルは特にPHPエクステンションの場所を定義するために使用できます。
*php.ini*初期化ファイルはデータベースの**Resources**フォルダーに配置します。最初の呼び出し時にこのファイルが存在しないと4Dは適切な設定オプションを使用してファイルを作成します。

外部インタープリターのphp.iniファイルは以下のエントリーを含んでいなければなりません:

- **auto_prepend_file**: *4D_Execute_PHP.php*ユーティリティスクリプトへの完全パス名を指定します。このスクリプトは[4D application]Resources/php/Windowsまたは/Macにあります。このエントリーがないと完全なスクリプトしか実行できません。スクリプト内のルーチン呼び出しが動作しなくなります。
- **display_errors**: PHPコード実行時に発生したエラーが4Dに通知されるようにするための"stderr"を設定します。例えば:

```
; stderr - Display the errors to STDERR (only affects the CGI/CLI)
; To direct the errors to STDERR for the CGI/CLI:
display_errors = "stderr"
```

カスタムphp.iniファイルの設定に関する情報は、4Dが提供するphp.iniファイルのコメントを参照してください。

PHP Execute

PHP Execute (scriptPath {; functionName {; phpResult {; param} {; param2 ; ... ; paramN}}}) -> 戻り値

引数	型		説明
scriptPath	テキスト	→	PHPスクリプトへのパスまたは "" でPHP関数を実行
functionName	テキスト	→	実行するPHP関数
phpResult	演算子, 変数, フィールド	←	PHP関数実行結果または結果を受け取らない場合*
param	テキスト, ブール, 実数, 倍長整数, 日付, 時間	→	PHP関数の引数
戻り値	ブール	↩	True = 正しく実行された False = 実行時にエラーがあった

説明

PHP ExecuteコマンドはPHPスクリプトや関数を実行するために使用します。.

scriptPath引数には、実行するPHPスクリプトのパス名を渡します。ファイルがデータベースストラクチャと同階層に存在する場合、相対パス名を指定できます。そうでなければ完全パスです。パス名はシステムシンタックスあるいはPOSIXシンタックスで表現できます。
標準のPHP関数を直接実行したい場合は、scriptPathに空の文字列 ("") を渡します。関数名は二番目の引数に渡さなければなりません。

scriptPathスクリプト内の特定の関数を実行したい場合、functionName引数にPHP関数名を渡します。この引数に空の文字列を渡したりfunctionName引数を省略した場合、スクリプト全体が実行されます。

Note: PHPの関数名は大文字小文字を区別します。括弧は使用せず、関数名のみを入力してください。

phpResult引数はPHP関数の実行結果を受け取ります。以下のいずれかを渡せます:

- 結果を受け取る変数、配列、またはフィールド
- 関数が結果を返さないか、結果を受け取る必要がない場合、* 文字。

phpResultにはテキスト、倍長整数、実数、ブール、または日付型、および (配列を除く) BLOBや時間型のフィールドを渡すことができます。4Dは後述の返されるデータの変換で説明する原則に基づき、データの変換と必要な調整を実行します。

- functionName引数に関数名を渡すと、phpResultはPHPの開発者が関数のコードから**return**コマンドで返す値を受け取ります。
- functionName引数に関数名を渡さずにコマンドを使用した場合、phpResultはPHPの開発者が**echo**や類似のコマンドで返す値を受け取ります。

引数を期待するPHP関数を呼び出す場合、1つ以上の値を渡すためにparam1...Nを使用します。値はセミコロンで分けられなければなりません。文字、テキスト、ブール、実数、整数、倍長整数、日付、時間タイプの値を渡すことができます。ピクチャとBLOBは渡せません。配列を送信することができます。この場合、PHP Executeコマンドには配列へのポインタを渡さなければなりません。そうしない場合、配列のカレントのインデックスが整数として送信されます (例題参照)。コマンドはポインタ、ピクチャおよび2D配列を除き、すべてのタイプの配列へのポインタを受け入れます。
param1...N引数はUTF-8のJSONフォーマットでPHPに送信されます。これらの引数はPHPのfunctionName関数に渡される前に、PHPの**json_decode**コマンドで自動でデコードされます。

注: 技術的な理由で、FastCGIプロトコル経由で渡す引数のサイズは64KBを超えてはなりません。テキスト型の引数を使用する際にはこの制限を考慮にいれる必要があります。

4D側でコマンドが正しく実行できると、言い換えれば実行環境の起動、スクリプトのオープン、そしてPHPインタープリターとの通信に成功すると、コマンドからTrueが返されます。そうでない場合、ON ERR CALLでとらえることができ、GET LAST ERROR STACKで解析できるエラーが生成されます。
さらにスクリプト自身がPHPエラーを生成するかもしれません。この場合PHP GET FULL RESPONSEコマンドを使用してエラーの発生元を解析しなければなりません (例題4参照)。

注: PHPを使用してエラー管理を設定できます。詳細は例えば以下のページを参照してください:
http://www.php.net/manual/en/errorfunc.configuration.php#ini.error-reporting.

返されるデータの変換

以下の表はphpResult引数の型に基づき、返されるデータが4Dによりどのように解釈され変換されるかを説明しています。

***phpResult*引数の型**	**4Dによる処理**	**例**
BLOB	4Dは受信したデータを変更せずに取り出します(*)。	
テキスト	4DはUTF-8でエンコードされたデータを期待します (*)。PHPの開発者はPHPの**utf8_encode**コマンドを使用する必要があるかもしれません。	PHPスクリプトの例: `echo utf8_encode(myText)`
日付	4Dは (PHPでときにDATE_ATOMと呼ばれる) RFC 3339フォーマットの文字列として送信される日付を期待します。このフォーマットは"YYYY-MM-DDTHH:MM:SS"という形式で、例えば2005-08-15T15:52:01+00:00のようになります。4Dは時間部を無視し、UTCの日付を返します。	
Time	4DはRFC 3339フォーマットの文字列として送信される時間を期待します (日付型参照)。4Dは日付部を無視し、ローカルタイムゾーンの日付を考慮した上で、0時からの経過秒数を返します。	2時間30分45秒を送信するPHPスクリプトの例: `echo date( DATE_ATOM, mktime( 2,30,45))`
整数または実数	4Dは数字や+、-、およびeに続く指数で表現された数値を解釈します。'.'や','文字はすべて小数区切り文字として解釈されます。	PHPスクリプトの例: `echo -1.4e-16;`
ブール	PHPから文字列"true"または非ヌルと数値評価される値を受け取ると4DはTrueを返します。	PHPスクリプトの例: `echo (a==b);`
配列	4Dは、PHP配列がJSONフォーマットで返されるものと見なします。	2つのテキストを返すPHPスクリプトの例: `echo json_encode( array( "hello", "world"));`

(*) デフォルトでHTTPヘッダーは返されません:
- functionName引数に関数名を渡してPHP Executeを使用すると、phpResultにHTTPヘッダーが返されることはありません。ヘッダーはPHP GET FULL RESPONSEコマンドを使用してのみ取得できます。
- 関数名なし (functionName引数を省略するかからの文字列を渡す) でPHP Executeを使用するとき、PHP SET OPTIONコマンドを使用してPHP Raw resultオプションをTrueに設定することでHTTPヘッダーを返すことができます。

注: PHPを使用して大量のデータを取得する必要がある場合、通常関数の戻り値を使用するよりも、(**echo**や同等のコマンドを使用して) *stdOut*バッファーを経由した方が効率的です。詳細はPHP GET FULL RESPONSEコマンドの説明を参照してください。

環境変数の利用

SET ENVIRONMENT VARIABLEコマンドを使用してスクリプトが使用する環境変数を指定できます。**警告**: LAUNCH EXTERNAL PROCESSや**PHP Execute**呼出し後、一連の環境変数は消去されます。

特別な関数

4Dは以下の特別な関数を提供します:

- **quit_4d_php**: PHPインタープリターとそのすべての子プロセスを終了するために使用します。スクリプトを実行中の子プロセスが存在すると、インタープリターは終了せず、**PHP Execute**コマンドはFalseを返します。
- **relaunch_4d_php**: PHPインタープリターを再起動するために使用します。

PHP Executeから最初のリクエストが送信されると、インタープリターが自動で再起動されることに留意してください。

例題 1

"myPhpFile.php"スクリプトを関数指定なしで呼び出します。スクリプトは以下の通りです:

```
<?php
echo 'Current PHP version: ' . phpversion();
?>
```

以下の4Dコードを実行すると:

```
 C_TEXT($result)
 C_BOOLEAN($isOK)
 $isOK:=PHP Execute("C:\\php\\myPhpFile.php";"";$result)
 ALERT($Result)
```

"Current PHP version: 5.3"と表示されます。

例題 2

"myNewScript.php"内のmyPhpFunction関数を引数付きで呼び出します。スクリプトは以下の通りです:

```
<?php
// . . . PHP code. . .
function myPhpFunction($p1, $p2) {
    return $p1 . ' '. $p2;
}
// . . . PHP code.. . .
?>
```

関数を呼び出します:

```
 C_TEXT($result)
 C_TEXT($param1)
 C_TEXT($param2)
 C_BOOLEAN($isOk)
 $param1 :="Hello"
 $param2 :="4D world!"
 $isOk:=PHP Execute("C:\\MyFolder\\myNewScript.php";"myPhpFunction";$result;$param1;$param2)
 ALERT($result) // "Hello 4D world!"が表示される
```

例題 3

PHPインタープリターを終了します:

```
 $ifOk:=PHP Execute("";"quit_4d_php")
```

例題 4

エラー管理:

```
  // エラー管理メソッドをインストール
 phpCommError:="" // PHPErrorHandler内で更新される
 $T_saveErrorHandler :=Method called on error
 ON ERR CALL("PHPErrorHandler")</p><p> // スクリプトを実行
 C_TEXT($T_result)
 If(PHP Execute("C:\\MyScripts\\MiscInfos.php";"";$T_result))
  // エラーなし, $T_Resultには結果が返される
 Else
  // エラーが検知された, PHPErrorHandlerメソッドにより管理
    If(phpCommError="")
       ... // PHPエラー, PHP GET FULL RESPONSEを使用する
    Else
       ALERT(phpCommError)
    End if
 End if

  // エラー管理メソッドをアンインストール
 ON ERR CALL($T_saveErrorHandler)
```

PHP_errHandlerメソッドは以下の通りです:

```
 phpCommError:=""
 GET LAST ERROR STACK(arrCodes;arrComps;arrLabels)
 For($i;1;Size of array(arrCodes))
    phpCommError:=phpCommError+String(arrCodes{$i})+" "+arrComps{$i}+" "+
    arrLabels{$i}+"\r"
 End for
```

例題 5

実行前に4Dで動的にスクリプトを作成します:

```
 DOCUMENT TO BLOB("C:\\Scripts\\MyScript.php";$blobDoc)
 If(OK=1)
    $strDoc:=BLOB to text($blobDoc;UTF8 Text without length)

    $strPosition:=Position("function2Rename";$strDoc)

    $strDoc:=Insert string($strDoc;"_v2";Length("function2Rename")+$strPosition)

    TEXT TO BLOB($strDoc;$blobDoc;UTF8 Text without length)
    BLOB TO DOCUMENT("C:\\Scripts\\MyScript.php";$blobDoc)
    If(OK#1)
       ALERT("スクリプトの作成中にエラーが発生しました。")
    End if
 End if
```

その後スクリプトを実行します:

```
 $err:=PHP Execute("C:\\Scripts\\MyScript.php";"function2Rename_v2";*)
```

例題 6

日付と時間タイプの値を直接受け取ります。スクリプトは以下の通りです:

```
<?php
// . . . code php. . .
echo date(DATE_ATOM, mktime(1, 2, 3, 4, 5, 2009));
// . . . code php. . .
?>
```

4D側で日付を受け取ります:

```
 C_DATE($phpResult_date)
 $result :=PHP Execute("C:\php_scripts\ReturnDate.php";"";$phpResult_date)
  //$phpResult_date は !2009/05/04!</p><p>4D側で時間を受け取ります:

 C_TIME($phpResult_time)
 $result :=PHP Execute("C:\php_scripts\ReturnDate.php";"";$phpResult_time)

  //$phpResult_time は ?01 :02 :03?
```

例題 7

配列にデータを配分します:

```
 ARRAY TEXT($arText ;0)
 ARRAY LONGINT($arLong ;0)
 $p1 :=","
 $p2 :="11,22,33,44,55"
 $phpok :=PHP Execute("";"explode";$arText;$p1;$p2)
 $phpok :=PHP Execute("";"explode";$arLong;$p1;$p2)

  // $arTextには文字値 "11", "22", "33",... が格納されます。
  // $arLongには数値 11, 22, 33,... が格納されます。
```

例題 8

配列を初期化します:

```
 ARRAY TEXT($arText ;0)
 $phpok :=PHP Execute("";"array_pad";$arText;->$arText;50;"undefined")
  // PHPで以下を実行: $arrTest = array_pad($arrTest, 50, ’undefined’);
  // $arTextは50 要素の"undefined"で埋められます。
```

例題 9

配列を使用して引数を渡します:

```
 ARRAY INTEGER($arInt;0)
 $phpok :=PHP Execute("";"json_decode";$arInt;"[13,51,69,42,7]")
  // PHPで以下を実行: $arInt = json_decode(’[13,51,69,42,7]’);
  // 配列に初期値が設定されます
```

PHP GET FULL RESPONSE

PHP GET FULL RESPONSE (stdOut {; errLabels ; errValues} {; httpHeaderFields {; httpHeaderValues}})

引数	型		説明
stdOut	テキスト変数, BLOB変数	←	stdoutバッファーの内容
errLabels	テキスト配列	←	エラーのラベル
errValues	テキスト配列	←	エラーの値
httpHeaderFields	テキスト配列	←	HTTPヘッダーの名前
httpHeaderValues	テキスト配列	←	HTTPヘッダーの値

説明

PHP GET FULL RESPONSE コマンドを使用して PHPインタープリターから返されるレスポンスに関する追加の情報を取得できます。このコマンドは特にスクリプトの実行中にエラーが発生したときに有効です。

PHPスクリプトはstdoutバッファーにデータを書き込むことがあります (echo, print等)。このコマンドはデータを直接stdout変数に返します。そしてPHP Executeコマンドで説明されている原則と同じ変換を適用します。

同期されるerrLabelsとerrValuesテキスト配列は、PHPスクリプトの実行がエラーの原因であるときに値が返されます。これらの配列には特に、エラーのもと、スクリプト、そして行などの情報が提供されます。これら2つの配列はともに使用します。errLabelsを渡すときは合わせてerrValuesも渡さなければなりません。

4DとPHP間の交換はFastCGI経由で実行されるため、PHPインタープリターは、それがHTTPサーバから呼び出されたかのように機能し、したがってHTTPヘッダを送信します。httpHeaderFieldsとhttpHeaderValues配列を使用してこれらのヘッダを取得できます。

PHP GET OPTION

PHP GET OPTION (option ; value)

引数	型		説明
option	倍長整数	→	取得するオプション
value	テキスト, ブール	←	オプションの現在値

説明

PHP GET OPTION コマンドを使用して、PHPスクリプトの実行に関連するオプションの現在値を取得できます。

option引数には取得するオプションを指定する"PHP"テーマの定数を渡します。コマンドはvalue引数にオプションの現在の値を返します。以下のいずれかの定数を渡すことができます:

定数	型	値	コメント
PHP Privileges	倍長整数	1	スクリプト実行に関連して指定されるユーザ権限定義 **とりうる値**: 以下の形式の文字列 "User:Password"。例: "root:jd51254d"
PHP Raw result	倍長整数	2	結果がテキスト型のときに実行結果中にPHPから返されるHTTPヘッダに関する処理モードの定義 (結果がBLOB型のときヘッダは常に保持されます)。 **とりうる値**: ブール。False (デフォルト値) = HTTPヘッダを結果から取り除く、True = HTTPヘッダを保持する

Note: PHP GET OPTIONコマンドでPHP Privilegesオプションを使用すると、ユーザアカウントのみが返され、パスワードは返されません。

例題

現在のユーザアカウントを知りたい場合:

```
C_TEXT($userAccount)
PHP GET OPTION(PHP Privileges;$userAccount)
ALERT($userAccount)
```

PHP SET OPTION

PHP SET OPTION (option ; value {; *})

引数	型		説明
option	倍長整数	→	設定するオプション
value	テキスト, ブール	→	オプションの新しい値
*	演算子	→	指定時: 変更は次の呼び出し時にのみ適用

説明

PHP SET OPTION コマンドを使用して、PHP Executeコマンド呼び出し前に、特定のオプションを設定することができます。このコマンドのスコープはカレントプロセスです。

option引数には、変更するオプションを指定する"PHP"テーマの定数を渡します。value引数にはoptionの新しい値を渡します。optionの説明は以下の通りです:

定数	型	値	コメント
PHP Privileges	倍長整数	1	スクリプト実行に関連して指定されるユーザ権限定義 **とりうる値**: 以下の形式の文字列 "User:Password"。例: "root:jd51254d"
PHP Raw result	倍長整数	2	結果がテキスト型のときに実行結果中にPHPから返されるHTTPヘッダに関する処理モードの定義 (結果がBLOB型のときヘッダは常に保持されます)。 **とりうる値**: ブール。False (デフォルト値) = HTTPヘッダを結果から取り除く、True = HTTPヘッダを保持する

デフォルトでPHP SET OPTIONはプロセス内で後に続くすべてのPHP Executeのオプションを設定します。次の呼び出しにのみ有効なオプションを設定するためには、アスタリスク (*) 引数を渡します。

例題

Adminアクセス権で"myAdminScript.php"スクリプトを実行します:

```
PHP SET OPTION(PHP Privileges;"admin:mypwd";*)
 // *を渡すので、admin権限は一回のみ使用される
C_TEXT($result)
C_BOOLEAN($isOK)
$isOK:=PHP Execute("myAdminScript.php";$result)
If($isOK)
   ALERT($result)
End if
```

PHPモジュールのサポート

このappendixでは、4DのPHPモジュール実装について説明します。以下の項目について説明します:

- 4DのPHPインタプリタでデフォルトで提供される標準PHPモジュールのリスト
- 4Dが保持しない標準PHPモジュールのリスト
- 追加モジュールのインストール

デフォルトで提供されるモジュール

以下の表は4Dにデフォルトで提供されるPHPモジュールの詳細です。

汎用モジュール

名称	Webサイト	説明
BCMath	http://php.net/bc	文字列として表された任意の大きさおよび 精度の数をサポートするバイナリ計算機

例:

```
C_LONGINT($value;$result)
$value:=4
$ok:=PHP Execute("";"bcpow";$result;$value;3)
```

カレンダー	http://php.net/calendar	異なったカレンダーフォーマット間の変換を 簡単に行う関数の集まり。ユリウス積算日を標準とする。

例:

```
C_LONGINT($NumberOfDays)
$ok:=PHP Execute("";"cal_days_in_month";$NumberOfDays;1;2;2010)
```

Ctype	http://php.net/ctype	現在のロケールに基づき文字または文字列がある文字クラスに含まれるかどうかを調べる関数。

例:

```
  // 提供された文字列のすべての文字が句読点かチェックする
C_TEXT($myString)
$myString:=",.;/"
$ok:=PHP Execute("";"ctype_punct";$isPunct;$myString)
```

日付・時刻	http://php.net/datetime	PHPスクリプトを実行するサーバから日付と時間を取得する。

例:

```
  // ポルトガルのリスボンにおける日の出時刻を計算する。
  // 緯度: 38.4 North、
  // 経度: 9 West、
  // 天頂 ~= 90、
  // 時差: +1 GMT
C_TIME($SunriseTime)
$ok:=PHP Execute("";"date_sunrise";$SunriseTime;0;1;38,41;-9;90;1)
```

DOM (Document Object Model)	http://php.net/dom	PHP5のDOM API による XMLドキュメントの処理。
Exif(*)	http://php.net/exif	画像のメタデータを扱う。
ファイル情報 (*)	http://php.net/fileinfo	ファイルのcontent-typeとエンコーディングを推測する。
Filter	http://php.net/filter	安全でないソース、例えばユーザ入力などによるデータの検証や除去を行う。

例:

```
C_LONGINT($filterId)
C_TEXT($result)
$ok:=PHP Execute("";"filter_id";$filterId;"validate_email")
$ok:=PHP Execute("";"filter_var";$result;"hop@123.com";$filterId)
```

FTP (File Transfert Protocol)	http://php.net/ftp	FTPサーバへの詳細なアクセスを提供。
Hash	http://php.net/hash	メッセージダイジェストエンジン。さまざまなハッシュアルゴリズムを使用して、任意の長さのメッセージに対する直接的あるいは段階的な処理を可能とする。

例:

```
C_TEXT($md5Result)
$ok:=PHP Execute("";"md5";$md5Result;"this is my string to hash")
```

GD (Graphics Draw) ライブラリ	http://php.net/gd	画像処理。
Iconv	http://php.net/iconv	様々な文字セット間でのファイル変換。
JSON (JavaScript Object Notation)	http://php.net/json	JSONデータ交換形式の実装
LDAP	http://php.net/ldap	LDAPは、"ディレクトリサーバ" にアクセスするために使用されるプロトコル。ディレクトリとは、ツリー構造に情報を保持している特殊なデータベース。
LibXML	http://php.net/libxml	XML関数や定数のライブラリ
マルチバイト文字列	http://php.net/mbstring	複数バイト文字エンコーディングの処理や文字列の変換に使用できる、文字処理のための一連の関数。
OpenSSL	http://php.net/openssl	OpenSSL の関数を使用して署名の生成、そして、データのシール (暗号化)およびオープン(復号化)を行う。
PCRE (Perl Compatible Regular Expressions)	http://php.net/pcre	Perl 5と同じシンタックスおよび語義を使用する一連の正規表現関数。

例:

```
  // この例は文字列から不要なスペースを取り除きます。
C_TEXT($myString)
$myString:="foo   o   bar"
PHP Execute("";"preg_replace";$myString;"/\\s\\s+/";" ";$myString)
ALERT($myString)
  // $myStringからスペースの繰り返しが削除され"foo o bar"になる。
```

PDO (PHP Data Objects) (*)	http://php.net/pdo	データベースアクセスのインタフェース。データベース毎のPDOドライバが必要。
PDO_SQLITE (*)	http://php.net/pdo_sqlite	SQLite 3にPHPからのアクセスを可能にする、PHP Data Objects (PDO) インタフェースを実装したドライバ。
リフレクション	http://php.net/reflection	完全なリフレクションAPIで、クラス、インターフェイス、関数、メソッド、そしてエクステンションのリバースエンジニアリングを可能にする。
Phar (PHP Archive)	http://php.net/phar	PHP アプリケーション全体をひとつの "phar" (PHP Archive) ファイルにまとめてしまい、配布やインストールを容易にする。
Session	http://php.net/session	PHPセッションのサポート

例:

セッションはWebアプリケーションにおいて、それぞれのリクエスト間でコンテキストを保持するために使用されます。4DでPHP Executeを実行すると、PHPスクリプトはセッションを開始し、コンテキストとして保持すべき情報を$_SESSION配列に格納することが可能となります。PHPスクリプトがセッションを使用する場合、PHP GET FULL RESPONSEコマンドを使用してPHPから返されるセッションIDを取得し、PHP Executeを呼び出す都度、事前にSET ENVIRONMENT VARIABLEコマンドを使用してセッションIDを指定しなければなりません。

```
  // "PHP Execute with context" メソッド
If(<>PHP_Session#"")
   SET ENVIRONMENT VARIABLE("HTTP_COOKIE";<>PHP_Session)
End if
If(PHP Execute($1))
   PHP GET FULL
RESPONSE($0;$errorInfos;$errorValues;$headerFields;$headerValues)
   $idx:=Find in array($headerFields;"Set-Cookie")
   If($idx>0)
      <>PHP_Session:=$headerValues{$idx}
   End if
End if
```

SimpleXML	http://php.net/simpleXML	とても簡単かつ容易に使用できるツールで、XMLをプロパティや配列反復子で処理可能なオブジェクトに変換するために使用します。
ソケット	http://php.net/sockets	BSDソケットに基づくソケット通信機能の低レベルインタフェースを実装し、クライアントとしてだけでなく、ソケットサーバとして動作させることが可能となります。
SPL (Standard PHP Library)	http://php.net/spl	標準的な問題を解決するためのインターフェイスやクラスを集めたもの。
SQLite (*)	http://php.net/sqlite	SQLiteデータベースエンジン用の拡張。このエンジンは埋め込み可能。
SQLite3 (*)	http://php.net/sqlite3	SQLite version 3データベースをサポート。
Tokenizer	http://php.net/tokenizer	字句解析レベルの言語処理を行うことなく、PHPソースを解析/修正するツールを作成することを可能にする関数。
XML (eXtensible Markup Language)	http://php.net/xml	XMLドキュメントの解析。
XMLreader	http://php.net/xmlreader	プル型のXMLパーサー。
XMLwriter	http://php.net/xmlwriter	XML形式のストリームやファイルを生成。
Zlib	http://php.net/zlib	gzip (.gz) 圧縮ファイルの読み込みと書き出し。

例:

```
WEB GET HTTP HEADER($names;$values)
$pos:=Find in array($names;"Accept-Encoding")
If($pos>0)
   Case of
      :(Position($values{$pos};"gzip")>0)
         WEB SET HTTP HEADER("Content-Encoding: gzip")
         PHP Execute("";"gzencode";$html;$html)
      :(Position($values{$pos};"deflate")>0)
         WEB SET HTTP HEADER("Content-Encoding: deflate")
         PHP Execute("";"gzdeflate";$html;$html)
   End case
End if
WEB SEND TEXT($html)
```

Zip	http://php.net/zip	ZIP圧縮アーカイブやその中のファイルの読み込みと書き出し。

(*) 現時点の4Dでは、Windows上でこれらのモジュールを利用できません。

Windowsでのみ利用可能なモジュール

構造的な理由から、以下のPHPモジュールはWindows上でのみ利用可能です。

名称	Web サイト	説明
COM & .NET	http://php.net/com	COM (Component Object Model) はWindowsプラットフォーム上でアプリケーションやコンポーネントが通信を行うおもな方法です。さらに、4DはCOMレイヤからの.Netアセンブリのインスタンス化や作成をサポートします。
ODBC (Open DataBase Connectivity)	http://php.net/odbc	標準のODBCサポートに加え、PHPの統合ODBC関数により、独自のAPI実装にODBC APIの構文を利用したいくつかのデータベースへのアクセスが提供されます。
WDDX (Web Distributed Data eXchange)	http://php.net/wddx	プラットフォームにかかわらず、Webを利用したWebアプリケーション間のデータ交換を容易にします。

無効にされたモジュール

以下のPHPモジュールは4Dに実装されていません。一番右の列にその理由が記載されています:

名称	Web サイト	理由 - 代わりのソリューション
Mimetype	http://php.net/mime-magic	廃止 (廃止予定) - Fileinfoを使用。
POSIX (Portable Operating System Interface)	http://php.net/posix	廃止 (廃止予定)。
Regular Expression (POSIX Extended)	http://php.net/regex	廃止 (廃止予定) - PCREを使用。
Crack	http://php.net/crack	ライセンスの制限。
ffmpeg	http://ffmpeg-php.sourceforge.net/	ライセンスの制限 - LAUNCH EXTERNAL PROCESSで、コマンドラインでffmpegを使用
Image Magick	http://php.net/manual/book.imagick.php	ライセンスの制限 - GD 2を使用。
IMAP (Internet Message Access Protocol)	http://php.net/imap	ライセンスの制限 - 4D Internet Commandsに統合されたコマンドを使用。
PDF (Portable Document Format)	http://php.net/pdf	ライセンスの制限 - Haru PDFを利用。
Mysqlnd (MySQL Native Driver)	http://dev.mysql.com/downloads/connector/php-mysqlnd/	4D環境に適切でない

php.ini ファイルのカスタマイズ

変更すべき "php.ini"ファイル(後述)は、4Dアプリケーションの **Resources\php** フォルダか(デフォルトファイル)、データベースの **Resources** フォルダ内(カスタムファイル)にあります。これについての詳細は、4DでPHPスクリプトを実行する を参照して下さい。

警告: "php.ini" ファイルに変更を加える場合は、十分な注意と、PHPに対する十分な知識が必要になります。カスタムの php.ini ファイルに関する詳細な情報は、4Dによって提供されている php.ini ファイル内のコメントを参考にすることができます。

注: PHP プロセスにかかる時間が比較的長い(30秒を超える)場合、自動的に'timeout' エラーが4Dに返され、プロセスは失敗します。この場合、デフォルトの *timeout* の時間を設定することによってPHP の実行の時間を確保することができます。これには二通りのやり方があります:

- "php.ini" ファイル内の **max_execution_time** 変数を設定する方法(秒数の値を渡します)。警告:この設定は全てのスクリプトに影響します。
- 長い時間を必要とするプロセスのPHP実行スクリプト内にて **set_time_limit(nbSec)** コマンドを呼び出す方法。PHPスクリプトの実行に与えられる時間の最大値を *nbSec* に渡します。この方法は指定したスクリプトのみを変更するので、こちらの方が推奨されます。通常はセキュリティ上の理由からPHPスクリプトの *timeout* 値は低めに抑えるのがよいでしょう。

追加モジュールのインストール

PHPインタプリタに追加のモジュールをインストールすることが可能です。これによりデフォルトで提供されていない機能を使用できます。

以下のタイプの拡張をインストールできます:

- PECL (PHP Extension Community Library) 拡張
- PEAR (PHP Extension and Application Repository) フレームワークの拡張
- Zend フレームワークの拡張
- Symphony フレームワークの拡張
- JELIX フレームワークの拡張
- eZコンポーネント

各タイプのインストール情報は以下の通りです。

Note: 4D で提供されるPHPは以下の通りです:

- バージョン5.4.11
- WindowsとMac OSで、32-bitコンパイルおよびスレッドセーフモード

Extensions PECL

Web サイト: http://pecl.php.net
PECL拡張を追加するには:

1. 目的のPECL拡張をダウンロードしてビルドする。
   または
   PHP 5.3 VC9 Non Thread Safe Windowsバイナリパッケージからビルド済みの拡張を取り出す (http://windows.php.net/download/#php-5.3-nts-VC9-x86)。
2. extensionフォルダに拡張を追加する。
3. "php.ini"ファイルで拡張を有効にする。

警告: PECL Webサイトで利用可能な拡張が制限の緩いPHPライセンスであったとしても、そのうちのいくつかはよりライセンスに制限のあるライブラリを必要とする場合があります。

他のWebサイトでPHP拡張を入手できる場合もありますが、この場合PHPグループの検証によりに提供されるセキュリティ保証はありません。

PEAR 拡張

Web サイト: http://pear.php.net/
PEARは完全なオブジェクト指向のフレームワークです。
PEAR拡張を追加するには:

1. PEARパッケージをダウンロード (http://pear.php.net/package/PEAR/download) して、"pear"フォルダに解凍する。
2. この"pear"フォルダを"php.ini"ファイルで指定した"include_path"に追加する。

Zend 拡張

1. Zendフレームワークをダウンロード (http://framework.zend.com/download/latest)して、"zend"フォルダに解凍する。
2. この"zend"フォルダを"php.ini"ファイルで指定した"include_path"に追加する。
3. Zendフレームワークコンポーネントのドキュメントを読む: http://framework.zend.com/manual/en

Symphony 拡張

Web サイト: http://www.symfony-project.org
SymphonyはWebコントローラを含むWebアプリケーションのフレームワークです。

1. 以下のアドレスからフレームワークとサンドボックスをダウンロードし、解凍する: http://www.symfony-project.org/installation/1_2.

JELIX 拡張

1. JELIXフレームワークをダウンロード (http://jelix.org/articles/en/download/stable) し、解凍する。
2. 出来上がった"jelix"フォルダを"php.ini"ファイルで指定した"include_path"に追加する。
3. JELIXフレームワークコンポーネントのドキュメントを読む: http://jelix.org/articles/en/manual-1.1/components

eZ components

1. eZ Componentsをダウンロード (http://www.ezcomponents.org/download) し、"ez"フォルダに解凍する。
2. 出来上がった"ez"フォルダを"php.ini"ファイルで指定した"include_path"に追加する。
3. eZ Componentsのドキュメントを読む: http://www.ezcomponents.org/docs/api/latest

SQL

- SQLコマンドの概要
- On SQL Authenticationデータベースメソッド
- Begin SQL
- End SQL
- Get current data source
- GET DATA SOURCE LIST
- Is field value Null
- QUERY BY SQL
- SET FIELD VALUE NULL
- SQL CANCEL LOAD
- SQL End selection
- SQL EXECUTE
- SQL EXECUTE SCRIPT
- SQL EXPORT DATABASE
- SQL EXPORT SELECTION
- SQL GET LAST ERROR
- SQL GET OPTION
- SQL LOAD RECORD
- SQL LOGIN
- SQL LOGOUT
- SQL SET OPTION
- SQL SET PARAMETER
- START SQL SERVER
- STOP SQL SERVER
- USE EXTERNAL DATABASE
- USE INTERNAL DATABASE

SQLコマンドの概要

4D には SQL カーネルが統合されています。さらに 4D には SQL サーバー機能が含まれていて、他の4Dアプリケーションや (4D ODBC Driver 経由で) サードパーティアプリケーションからクエリを実行できます。

4DのSQL関連ドキュメントは大きく二つあります:

- **4D SQL リファレンスガイド** (SQLリファレンス): このマニュアルでは4DのSQLカーネルにアクセスするための様々な方法、SQLサーバー設定、およびSQLクエリで使用できるコマンドやキーワード (例えば SELECT や UPDATE) について説明しています。4Dに実装されたSQLランゲージについて知りたい場合はこのマニュアルを参照してください。
- **4D ランゲージリファレンスの "SQL" テーマ** (SQL): このテーマには、4DランゲージからSQLを利用するためのハイレベルコマンドがまとめられています:
  - SQL サーバーの制御: START SQL SERVER と STOP SQL SERVER
  - 統合 SQL カーネルへの直接アクセス: SET FIELD VALUE NULL, Is field value Null, QUERY BY SQL
  - 外部または内部データソース (SQL パススルー) への接続管理: GET DATA SOURCE LIST, Get current data source, SQL LOGIN, SQL LOGOUT
  - 直接SQL接続またはODBCを経由した接続のフレームワークにおけるデータ処理ハイレベルコマンド: Begin SQL, End SQL, SQL CANCEL LOAD, SQL LOAD RECORD, SQL EXECUTE, SQL End selection, SQL SET OPTION, SQL SET PARAMETER, SQL GET LAST ERROR, SQL GET OPTION.

ハイレベルSQLコマンドの動作

4Dの組み込みSQLコマンドは、接頭辞"SQL" で始まります。そして、以下の原則が実装されます:

- 特に断らない限り、これらのコマンドを4Dの内部SQLカーネル、外部へのダイレクトまたはODBC経由の接続で使用できます。SQL LOGINコマンドで接続方法を指定できます。
- 接続のスコープはプロセスです。同時に複数の接続を実行したい場合、新たに開始したプロセスでSQL LOGINを実行します。
- ON ERR CALLコマンドを使用して、ハイレベルSQLコマンドを実行している間に発生したODBCエラーを捕らえることができます。追加情報を得るにはSQL GET LAST ERRORコマンドを使用します。

標準ODBCのサポート

ODBC (Open DataBase Connectivity) 標準は、標準化された機能のライブラリを定義します。これらの機能により4Dのようなアプリケーションは、SQLランゲージでODBC互換のデータ管理シ ステム(データベース、スプレッドシート、他の4Dのアプリケーションなど) へアクセスできます。

注: 4DではIMPORT ODBCやEXPORT ODBCコマンドを使用して、あるいはデザインモードで手動により、ODBCソースからデータを読み込んだり、ODBCソースへデータを書き出したりすることが可能です。詳細については、*4D Design Reference*マニュアルを参照してください。

注: 4DのハイレベルSQLコマンドを使用して、ODBCデータソースと通信する単純な4Dアプリケーションを実装できます。もしアプリケーションでODBC標準の広範なサポートが必要である場合、4DのローレベルODBCプラグインである、**4D ODBC Pro**をお使いください。

データタイプの対応

以下は4Dが自動的に設定する4DのタイプとSQLデータタイプの対応リストです:

4Dのタイプ	SQLのタイプ
C_STRING	SQL_C_CHAR
C_TEXT	SQL_C_CHAR
C_REAL	SQL_C_DOUBLE
C_DATE	SQL_C_TYPE_DATE
C_TIME	SQL_C_TYPE_TIME
C_BOOLEAN	SQL_C_BIT
C_INTEGER	SQL_C_SHORT
C_LONGINT	SQL_C_SLONG
C_BLOB	SQL_C_BINARY
C_PICTURE	SQL_C_BINARY
C_GRAPH	SQL_C_BINARY

SQLリクエストで4D式を参照する

SQLリクエストに4D式 (変数、配列、フィールド、ポインタ、有効な式) を挿入するには2通りの方法があります。一つは直接割り当てる方法。もう一つはSQL SET PARAMTERで引数を設定する方法です。

直接の割り当ては2つの方法で実行できます:

- リクエストのテキスト中で、4Dオブジェクト名を<<と >>文字の間に挿入する
- 参照をコロンの後に記述する

```
SQL EXECUTE("INSERT INTO emp (empnum,ename) VALUES (<<vEmpnum>>,<<vEname>>)")
SQL EXECUTE("SELECT age FROM People WHERE name= :vName")
```

Note: コンパイルモードでは、($で始まる) ローカル変数への参照を使用することはできません。

この例では、リクエスト実行時に引数が4DのvEmpnum、vEnameおよびvNametheの現在値に置き換えられます。このソリューションは4Dフィールドと配列でも動作します。

この簡単に使用できるシンタックスには、SQL標準に対応していない、また出力引数の使用を許可しないといった欠点もあります。これを修正するには、SQL SET PARAMETERコマンドを使用します。このコマンドを使用して使用モード (入力と出力) とともに、リクエストに各4Dオブジェクトを統合できます。作り出されたシンタックスは標準のものです。詳細については、SQL SET PARAMETERコマンドの説明を参照してください。

1. この例題では4D配列を直接関連付けてSQLクエリを実行します:

```
ARRAY TEXT(MyTextArray;10)
ARRAY LONGINT(MyLongintArray;10)

For(vCounter;1;Size of array(MyTextArray))
   MyTextArray{vCounter}:="Text"+String(vCounter)
   MyLongintArray{vCounter}:=vCounter
End for
SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (<<MyTextArray>>, <<MyLongintArray>>)"
SQL EXECUTE(SQLStmt)
```

2. この例題では4Dフィールドを直接関連付けて、SQLクエリを実行します:

```
 ALL RECORDS([Table_2])
 SQL LOGIN("mysql";"root";"")
 SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES
(<<[Table_2]Field1>"+">,<<[Table_2]Field2>>)"
 SQL EXECUTE(SQLStmt)
```

3. この例題では逆参照されたポインタを使用して直接変数を渡して、SQLクエリを実行します:

```
C_LONGINT($vLong)
C_POINTER($vPointer)
$vLong:=1
$vPointer:=->$vLong
SQL LOGIN("mysql";"root";"")
SQLStmt:="SELECT Col1 FROM TEST WHERE Col1=:$vPointer"
SQL EXECUTE(SQLStmt)
```

コンパイルモードでローカル変数を使用する

コンパイルモードでは特定の条件下において、SQL文の中でローカル変数参照を使用できます:

- Begin SQL / End SQLブロック内でローカル変数を使用できます。ただしEXECUTE IMMEDIATEコマンドは除きます;
- 変数を (参照ではなく) 引数内で直接使用する場合、SQL EXECUTEコマンドでローカル変数を使用できます。
  例えば以下のコードはコンパイルモードで動作します:
  ```
  SQL EXECUTE("select * from t1 into :$myvar")
  ```
  以下のコードはコンパイルモードでエラーを生成します:
  ```
  C_TEXT(tRequest)
  tRequest:="select * from t1 into :$myvar"
  SQL EXECUTE(tRequest)
  ```

4Dで値を受け取る

4Dランゲージ中でSQLクエリの値を受け取る方法は2つあります:

- SQL EXECUTE コマンドの追加の引数を使用する (推奨)
- SQLクエリのINTO節を使用する (特別な場合のために予約されたソリューション)

リストボックスにSQLクエリの結果を表示する

SQLクエリーの結果を直接配列タイプのリストボックスに表示することができます。これによりSQLクエリーの結果を素早く見る方法が提供されます。SELECTタイプのクエリーのみを使用できます。このメカニズムは外部SQLデータベースには使用できません。

この機能は以下の原則に基づいて動作します:

- クエリーの結果を受け取るリストボックスを作成します。リストボックスのデータソースは**配列**に設定しなければなりません。
- SELECTタイプのSQLクエリーを実行し、結果をリストボックスに割り当てた変数に受け取ります。Begin SQL/End SQL キーワードを使用できます (4Dランゲージリファレンス参照)。
- ユーザーはリストボックス列をソートしたり、更新したりできます。
- SELECTクエリーを実行するたびに、リストボックス列はリセットされます (複数のSELECTクエリーを実行して、リストボックスに行を追加することはできません)。
- SQLクエリーの結果受け取る列数と同じ数の列をリストボックスに用意することを推奨します。SELECTクエリーにより返される列数よりもリストボックスの列数が少ない場合、自動で列が追加されます。SELECTクエリーの結果よりも多い列数がある場合、不必要な列は自動で隠されます。
  注: 自動で追加された列には一時的な配列がバインドされます。この一時的な配列はフォームが閉じられるまで存在します。また一時的な変数が各ヘッダー用に作成されます。LISTBOX GET ARRAYSコマンドが呼び出されると、*arrColVars*引数には一時的な配列へのポインターが、*arrHeaderVars*引数には一時的なヘッダー変数へのポインターが返されます。例えば5つの列が追加されると、5番目の列の配列名は*sql_column5*でヘッダー名は*sql_header5*となります。
- インタープリターモードでは、SQLクエリーにより返されたデータに基づき、自動でリストボックスの既存の配列が型変更される場合があります。

例
PEOPLEテーブルのすべてのフィールドのデータを取得し、*vlistbox*という変数名のリストボックスに結果を表示します。これを行うためのメソッドは以下の通りです:

```
Begin SQL
   SELECT * FROM PEOPLE INTO <<vlistbox>>
End SQL
```

On SQL Authenticationデータベースメソッド

On SQL Authenticationデータベースメソッドは4Dに統合されたSQLサーバへ送られたリクエストを選別します。この選別は、名前とパスワード、そしてユーザのIPアドレス (オプション) に基づいて実行されます。開発者は独自のユーザーテーブルや、4Dのユーザーテーブルを使用して、接続を識別できます。接続を認証したら、CHANGE CURRENT USER コマンドを呼び出して、4Dのデータベース内のリクエストへのアクセスをコントロールしなければなりません。

On SQL Authenticationデータベースメソッドが存在する場合、4Dまたは4D ServerのSQLサーバに外部からSQL接続が行われると、自動的にこのメソッドが呼び出されます。4Dユーザを管理する内部システムは起動しません。データベースメソッドが$0にTrueを返しかつ、CHANGE CURRENT USERコマンドの実行が成功した場合のみ、接続が受け入れられます。これらの条件を満たさない場合リクエストは拒否されます。

Note: SQL LOGIN(SQL_INTERNAL;$user;$password)ステートメントは内部接続となるため、On SQL Authenticationデータベースメソッドを呼び出しません。

データベースメソッドは最大3つのテキストタイプの引数を4Dより受け取り、$0にブール値を返します。以下はこれらの引数の説明です。

引数	型	説明
$1	テキスト	ユーザ名
$2	テキスト	パスワード
$3	テキスト	(オプション) リクエスト元のクライアントのIPアドレス
$0	ブール	True = リクエストを許可, False = リクエストを拒否

以下のようにこれらの引数を宣言しなければなりません:

```
  ` On Web Authentication データベースメソッド

 C_TEXT($1;$2;$3)
 C_BOOLEAN($0)
  ` メソッドコード
```

標準テキストとして、パスワード($2)を受け取ります。

On SQL AuthenticationデータベースメソッドでSQL接続の識別子を確認します。例えば、ユーザのカスタムテーブルを使用して名前とパスワードをチェックします。識別子が有効な場合は、$0にTrueを返して接続を受け入れます。その他の場合は$0に、Falseを返して接続が拒否されます。

Note: On SQL Authenticationデータベースメソッドが存在しない場合、4Dの統合されたユーザ管理システムを使用して、接続を決定します (有効になっている場合、つまりDesignerにパスワードが割り当てられている場合)。このシステムが起動していない場合、ユーザはDesignerアクセス権 (フリーアクセス) で接続されます。

$0にTrueを渡す場合、リクエストを受け入れ、ユーザのためにSQLのセッションを開くためには、**On SQL Authenticationデータベースメソッド**でCHANGE CURRENT USER コマンドを呼び出し、その実行が成功しなければなりません。

CHANGE CURRENT USER コマンドは、仮想の認証システムを実行するために使用されます。この認証システムには、2つの利点があります。1つは接続動作をコントロールできること。もう1つは4DのSQLセッションで接続の識別子を外部から見えないようにします。

以下の例で**On SQL Authenticationデータベースメソッド**は、接続リクエストが内部ネットワークからのものであることを確認し、識別子を検証、SQLセッションへのアクセス権に"sql_user" ユーザを割り当てます。

```
 C_TEXT($1;$2;$3)
 C_BOOLEAN($0)
  `$1: user
  `$2: password
  `($3: IP address of client)
 ON ERR CALL("SQL_error")
 If(checkInternalIP($3))
  `checkInternalIPメソッドはIPアドレスが内部のものか確認します。
    If($1="victor") & ($2="hugo")
       CHANGE CURRENT USER("sql_user";"")
       If(OK=1)
          $0:=True
       Else
          $0:=False
       End if
    Else
       $0:=False
    End if
 Else
    $0:=False
 End if
```

Begin SQL

Begin SQL
このコマンドは引数を必要としません

説明

Begin SQLはメソッドエディタで使用するキーワードで、プロセスのカレントデータソース (4Dの統合SQLエンジン、またはSQL LOGINコマンドで特定されたソース) により解釈されるべき一連のコマンドの始まりを宣言します。

Begin SQLで開始された一連のSQLコマンドは、End SQLキーワードで閉じなければなりません。

これらのキーワードは以下のように動作します:

- 同じメソッドに、一つ以上の**Begin SQL/End SQL**タグのブロックを置くことができ、すべてSQLコードから成るメソッドや4DコードとSQLコードを混合させたメソッドも作成することができます。
- 同じ行に幾つかの SQLステートメント を書き込み、それらのSQLステートメントをセミコロン ";" で区切ることもできます。例えば、以下のように書きこむことができます。

```
Begin SQL
   INSERT INTO SALESREPS (NAME, AGE) VALUES (Henry,40);
   INSERT INTO SALESREPS (NAME, AGE) VALUES (Bill,35)
End SQL
```

または

```
Begin SQL
   INSERT INTO SALESREPS (NAME, AGE) VALUES (Henry,40);INSERT INTO SALESREPS (NAME, AGE) VALUES (Bill,35)
End SQL
```

4Dのデバッガは行ごとにSQL命令行を評価します。一行以上使用した方が好ましい場合もありますのでご注意下さい。

End SQL

End SQL
このコマンドは引数を必要としません

説明

End SQLとはメソッドエディタ中で一連のSQLコマンドの最後を意味するキーワードです。

一連のSQLステートメントをBegin SQLとEnd SQL キーワードで囲みます。詳細については、Begin SQLキーワードの説明を参照してください。

Get current data source

Get current data source -> 戻り値

引数	型	説明
戻り値	文字	← 使用されているカレントのデータソース名

説明

Get current data sourceコマンドはアプリケーションのカレントデータソースの名前を返します。カレントデータソースは、Begin SQL/End SQL内で実行されるSQLクエリを受け取ります。
カレントデータソースがローカルな4Dのデータベースである時、コマンドはSQL_INTERNAL定数("" テーマ) の値に該当する文字列";DB4D_SQL_LOCAL;" を返します。
SQLクエリを実行する前に、このコマンドでカレントデータソースをチェックすることができます。

GET DATA SOURCE LIST

GET DATA SOURCE LIST (sourceType ; sourceNamesArr ; driversArr)

引数	型		説明
sourceType	倍長整数	→	ソースタイプ: ユーザまたはシステム
sourceNamesArr	テキスト配列	←	データソース名の配列
driversArr	テキスト配列	←	ソース用のドライバの配列

説明

GET DATA SOURCE LISTコマンドは、オペレーションシステムのODBCマネージャで定義されているsourceType データソースのドライバと名前をsourceNamesArrとdriversArrの配列に返します。

4Dではランゲージによる外部ODBCデータソースへのダイレクト接続が可能です。そしてBegin SQL/End SQLタグ構造内でSQLクエリを実行します。これは以下のように機能します。GET DATA SOURCE LISTコマンドはマシーン上に存在するデータソースのリストを得るために使用されます。SQL LOGINコマンドは使用するソースを指定するためのコマンドです。Begin SQL/End SQLタグストラクチャを使用してカレントソースにSQLクエリを実行することができます。4Dの内部エンジンを再度使用してクエリを実行するには、SQL LOGOUTコマンドを実行するだけです。メソッドエディタのSQLコマンドについての詳細は、4D SQL Reference マニュアルを参照してください。

sourceTypeには、検索したいデータソースのタイプを渡します。""テーマにある以下のいずれかの定数を使用できます。

定数	型	値
System Data Source	倍長整数	2
User Data Source	倍長整数	1

Note: このコマンドはファイル型データソースを除外します。

このコマンドは該当する値でsourceNamesArrとdriversArr配列に書き込み、サイズを調節します。

Note: ODBC経由で外部4Dデータソースへ接続したい場合、お手持ちのマシーンに 4DのODBC Driverをインストールしなければなりません。詳細については、4D ODBC Driver Installationマニュアルを参照してください。

例題

以下はユーザデータソースを使用する例です:

```
ARRAY TEXT(arrDSN;0)
ARRAY TEXT(arrDSNDrivers;0)
GET DATA SOURCE LIST(User Data Source;arrDSN;arrDSNDrivers)
```

システム変数およびセット

コマンドが正しく実行されるとOKシステム変数が1に設定されます。そうでなければ0が設定されエラーが生成されます。

Is field value Null

Is field value Null (aField) -> 戻り値

引数	型		説明
aField	フィールド	→	評価するフィールド
戻り値	ブール	⮐	True = フィールドはNULL, False = フィールドはNULLでない

説明

Is field value NullコマンドはaField引数 によって指定されたフィールドがNULL値を含む場合Trueを返します。その他の場合Falseを返します。

4DのSQLカーネルはNULL値を使用します。詳細については、4DのSQL Referenceマニュアルを参照して下さい。

QUERY BY SQL

QUERY BY SQL ({aTable ;} sqlFormula)

引数	型	説明
aTable	テーブル	→ レコードセレクションを返すテーブル、または 省略された場合デフォルトテーブル
sqlFormula	文字	→ SELECTクエリのWHERE節を表す 有効なSQL検索フォーミュラ

説明

4Dに統合されたSQLカーネルのメリットを利用ためにQUERY BY SQLコマンドを使用できます。このコマンドで以下のような簡単なSELECTクエリを実行できます:

```
SELECT *
   FROM aTable
   WHERE <sqlFormula>
```

aTableは、最初の引数に渡されるテーブルの名前です。sqlFormulaは、2番目の引数で渡されるクエリの文字列です。

例えば、以下のステートメントは、

```
 ([Employees];"name='smith'")
```

以下のSQLクエリに相当します。

```
SELECT * FROM Employees WHERE "name='smith'"
```

QUERY BY SQL コマンドは、QUERY BY FORMULA コマンドと類似しています。指定されたテーブルでレコードを探します。このコマンドは、カレントプロセスのaTableのカレントセレクションを変更し、新しいセレクションの最初のレコードをカレントレコードにします。

Note: QUERY BY SQL コマンドは、外部SQL接続のコンテクストでは使用されません。このリクエストは4Dの統合されたSQLエンジンに直接接続します。

QUERY BY SQLは、テーブルセレクションの各レコードにsqlFormulaを適用します。sqlFormulaはブール式で、TrueまたはFalseを返さなければなりません。SQL標準では、検索条件はTrue、FalseまたはNULLを返します。検索条件がTrueを返す全てのレコード (行) が、新しいカレントセレクションに含まれます。

例えば、値とフィールド (カラム) と比較する場合、sqlFormula式は単純です。しかし演算などを実行したりすると、sqlFormulaの式は複雑になります。QUERY BY FORMULAのように、QUERY BY SQLはリレートするテーブルの情報を評価できます (例題4を参照)。sqlFormulaは有効なSQLステートメントでなければなりません。そしてそのステートメントは4Dの現在のSQLの実行規則の点においてSQL-2スタンダードに準じていなければなりません。4DのSQLのサポートについては、4D SQL Referenceマニュアルを参照してください。

sqlFormula引数は、4D式への参照を使用できます。使用できるシンタックスは、統合SQLコマンドやBegin SQL/End SQLタグの間に挿入されるコード (<<MyVar>>または:MyVar) と同じです。詳細は、SQLコマンドの概要の節を参照してください。

注: このコマンドは、SET QUERY LIMITやSET QUERY DESTINATIONコマンドと互換です。

注: コンパイルモードではローカル変数への参照を使用することはできません。4DにおけるSQLプログラミングの情報はSQLコマンドの概要を参照してください。

リレーションについて

4Dのストラクチャエディタで定義されたテーブル間で、QUERY BY SQLはリレーションを使用しません。関連するデータを利用したい場合、クエリへJOINを追加する必要があります。例えば、[Persons]Cityから[Cities]Nameの間に、N対1リレーションを持つ以下のストラクチャがあると仮定します:

```
[People]
   Name
   City
[Cities]
   Name
   Population
```

QUERY BY FORMULAコマンドを使用して、以下のように記述できます:

```
 QUERY BY FORMULA([People];[Cities]Population>1000)
```

QUERY BY SQLを使用して、リレーションの存在の有無に関わらず、以下のステートメントを記述できます:

```
 QUERY BY SQL([People];"people.city=cities.name AND cities.population>1000")
```

Note: QUERY BY SQLは、QUERY BY FORMULAと異なる方法で、1対NとN対Nリレーションを扱います。

例題 1

売上高が100を超えるオフィスを表示します。SQLは以下のようになります:

```
SELECT *
   FROM Offices
   WHERE Sales > 100
```

QUERY BY SQL コマンドを使用すると、

```
 C_STRING(30;$queryFormula)
 $queryFormula:="Sales > 100"
 QUERY BY SQL([Offices];$queryFormula)
```

例題 2

3000から4000件の範囲に分類される注文を表示します。SQLは以下のようになります:

```
SELECT *
   FROM Orders
   WHERE Amount BETWEEN 3000 AND 4000
```

QUERY BY SQL コマンドを使用すると、

```
 C_STRING(40;$queryFormula)
 $queryFormula:="Amount BETWEEN 3000 AND 4000"
 QUERY BY SQL([Orders];$queryFormula)
```

例題 3

指定された条件で並び替えされたクエリ結果の取得方法を説明します。SQLは以下のようになります:

```
SELECT *
   FROM People
   WHERE City ='Paris'
      ORDER BY Name
```

QUERY BY SQL コマンドを使用すると、

```
 C_STRING(40;$queryFormula)
 $queryFormula:="City= 'Paris' ORDER BY Name"
 QUERY BY SQL([People];$queryFormula)
```

例題 4

4Dのリレートテーブルを使用するクエリをこの例で示します。SQLでは、JOINを使用してリレーションを表わします。以下の2つのテーブルがあると仮定します:

```
[Invoices] :
   ID_Inv: Longint
   Date_Inv: Date
   Amount: Real
[Lines_Invoices] :
   ID_Line: Longint
   ID_Inv: Longint
   Code: Alpha (10)
```

[Lines_Invoices]ID_Invから[Invoices]ID_Invの間に、N対1のリレーションがあります。
QUERY BY FORMULAコマンドでストラクチャのリレーションを使用する場合、以下のように記述します:

```
 QUERY BY FORMULA([Lines_Invoices];([Lines_Invoices]Code="FX-200") & (Month of([Invoices]Date_Inv)=4))
```

これをSQLクエリで表すと以下のようになります:

```
SELECT ID_Line
   FROM Lines_Invoices, Invoices
   WHERE Lines_Invoices.ID_Inv=Invoices.ID_Inv
      AND Lines_Invoices.Code='FX-200'
      AND MONTH(Invoices.Date_Inv) = 4
```

QUERY BY SQL コマンドを使用する場合:

```
 C_STRING(40;$queryFormula)
 $queryFormula:="Lines_Invoices.ID_Inv=Invoices.ID_InvAND Lines_Invoices.Code='FX-200' AND 
MONTH(Invoices.Date_Inv)=4"
 QUERY BY SQL([Lines_Invoices];$queryFormula)
```

システム変数およびセット

検索条件のフォーマットが正しければ、OKシステム変数は1に設定されます。そうでなければこのコマンドの結果のセレクションは空になり、エラーが返され、OK変数の値は0に設定されます。このエラーはON ERR CALLON ERR CALLコマンドでインストールされるエラー処理メソッドでとらえることができます。

SET FIELD VALUE NULL

SET FIELD VALUE NULL (aField)

引数	型		説明
aField	フィールド	⇒	NULL値を割り当てるフィールド

説明

SET FIELD VALUE NULLコマンドは、aField 引数によって指定されたフィールドにNULL値を割り当てます。

NULL値は、4DのSQLカーネルによって使用されます。詳細については、4D SQL Referenceマニュアルを参照して下さい。

Note: ストラクチャエディタレベルで、4DフィールドへのNULL値を非許可にすることができます(Design Referenceマニュアルを参照)。

SQL CANCEL LOAD

SQL CANCEL LOAD
このコマンドは引数を必要としません

説明

SQL CANCEL LOAD コマンドコマンドは、現在のSELECTリクエストを終了してパラメタを初期化します。

このコマンドを使用して、SQL LOGINコマンドにより開始された同一接続内（つまり同一カーソル内）において、複数のSELECTリクエストを実行できます。

例題

この例題では、同一接続内で2つのリクエストが実行されます:

```
C_BLOB(Myblob)
C_TEXT(MyText)
SQL LOGIN("mysql";"root";"")

SQLStmt:="SELECT blob_field FROM app_testTable"
SQL EXECUTE(SQLStmt;Myblob)
While(Not(SQL End selection))
   SQL LOAD RECORD
End while

  `カーソルを置きなおす。
SQL CANCEL LOAD

SQLStmt:="SELECT Name FROM Employee"
SQL EXECUTE(SQLStmt;MyText)
While(Not(SQL End selection))
   SQL LOAD RECORD
End while
```

システム変数およびセット

コマンドが正しく実行されると、システム変数OKは1に、そうでなければ0に設定されます。

SQL End selection

SQL End selection -> 戻り値

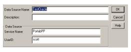
引数	型		説明
戻り値	ブール	↩	結果セットの境界に達した

説明

SQL End selectionコマンドは、結果セットの境界に達したかどうかを判定するために使用します。

例題

次のコードは、外部データソース(Oracle)へ接続します:



Data Source Name: TestOracle
Description:
Data Source
Service Name: PortabPP
UserID: scott

```
C_TEXT(vName)

SQL LOGIN("TestOracle";"scott";"tiger")
If(OK=1)
   SQL EXECUTE("SELECT ename FROM emp";vName)
   While(Not(SQL End selection))
      SQL LOAD RECORD
   End while
   SQL LOGOUT
End if
```

このコードは4D変数vName にempテーブルの ename を返します。

SQL EXECUTE

SQL EXECUTE (sqlStatement {; boundObj}{; boundObj2 ; ... ; boundObjN})

引数	型		説明
sqlStatement	テキスト	→	実行するSQLコマンド
boundObj	変数, フィールド	←	結果を受け取る(必要に応じて)

説明

SQL EXECUTEコマンドを使用してSQLコマンドを実行し、結果を4Dのオブジェクト(配列、変数またはフィールド) にバインドできます。

このコマンドを実行するためには、カレントプロセスで有効な接続が指定されていなければなりません。

sqlStatement引数には実行するSQLコマンドが含まれています。boundObjはその結果を受け取ります。変数は列の順番でバインドされ、残っているリモートカラムは放棄されます。

boundObjに4Dのフィールドが渡された場合、コマンドはレコードを作成し自動的に保存します。4Dのフィールドは同じテーブルに属していなければなりません (テーブル1のフィールドとテーブル2のフィールドを同じ呼び出しで渡すことはできません)。複数のテーブルのフィールドが渡された場合、エラーが発生します。

警告: boundObj 引数に4Dのフィールドを渡してSELECTコマンドを実行した場合、常にリモート4Dのデータが更新されます。リモートソースのデータをローカルに取得したい場合、まずデータを配列に取得してINSERTコマンドを呼び出します (例題 6参照)。

boundObj引数に4Dの配列を渡す場合、コマンドを呼び出す前に、その配列を宣言することをお勧めします。これは処理されるデータのタイプを検証するためです。必要に応じて配列は自動でサイズ変更されます。

4D変数の場合、1度に1つのレコードが取得されます。他の結果は無視されます。

注: SQLクエリで4D式を参照することについての詳細は、SQLコマンドの概要を参照してください。

例題 1

以下の例で、データソースにあるempテーブルのenameカラムを取得します。その結果は、4Dのフィールドの[Employee]Nameに保存されます。4Dのレコードは自動的に作成されます。

```
SQLStmt:="SELECT ename FROM emp"
SQL EXECUTE(SQLStmt;[Employee]Name)
SQL LOAD RECORD(SQL All Records)
```

例題 2

レコードの作成を確認するには、トランザクションにコードを投入します。そしてオペレーションが十分であると判明した場合のみ、それを認証します。

```
SQL LOGIN("mysql";"root";"")
SQLStmt:="SELECT alpha_field FROM app_testTable"
START TRANSACTION
SQL EXECUTE(SQLStmt;[Table2]Field1)
While(Not(SQL End selection))
   SQL LOAD RECORD
   ... `ここにデータを検証するコードを設定します。
End while
VALIDATE TRANSACTION `トランザクションの検証
```

例題 3

以下の例では、データソースにあるempテーブルのenameカラムを取得します。その結果は、aName 配列に保存されます。1度に10個のレコードを取って来ます。

```
ARRAY TEXT(aName;20)
SQLStmt:="SELECT ename FROM emp"
SQL EXECUTE(SQLStmt;aName)
While(Not(SQL End selection))
   SQL LOAD RECORD(10)
End while
```

例題 4

以下の例では、データソースの特定のID(WHERE節) のためのempテーブルのenameとジョブを取得します。その結果は、4Dの変数、vNameとvJobに保存されます。最初のレコードだけを取って来ます。

```
SQLStmt:="SELECT ename, job FROM emp WHERE id = 3"
SQL EXECUTE(SQLStmt;vName;vJob)
SQL LOAD RECORD
```

例題 5

以下の例では、データソースにあるTestテーブルのBlob_Fieldカラムを取得します。その結果は、BLOB変数に保存されます。そして、レコードがロードされる度に、そのBLOB変数の値を更新します。

```
C_BLOB(MyBlob)
SQL LOGIN
SQL EXECUTE("SELECT Champ_Blob FROM Test";MonBlob)
While(Not(SQL End selection))
 `結果を調査します。
   SQL LOAD RECORD
 `呼び出す度にMyBlobの値を更新します。
End while
```

例題 6

リモートの4D Serverデータベースからデータをローカルに取得します:

```
  // リモートデータベースにログインする
 SQL LOGIN("IP:192.168.18.15:19812";"user";"password";*)
 If(OK=1)
  // これ以降、すべてのSQL命令はリモートデータベース上で実行されます
    C_TEXT($LastName_value) // 検索文で使用される4D変数
    ARRAY TEXT($a_LastName;0) // LastNameデータの一時的な格納場所
    ARRAY TEXT($a_FirstName;0) // FirstNameデータの一時的な格納場所
    C_BOOLEAN($UseSQL) //データをローカルに格納する方法の選択

    $LastName_value:="Smith" // 4D変数の初期化

  // SQLリクエスト中の一番目のプレースホルダー "?" に、4Dの$LastName_value変数を割り当てる
    SQL SET PARAMETER($LastName_value;SQL Param In)

  // リモートのPERSONSテーブルから"LastName = Smith"であるLastNameとFirstNameカラムの
  // 値を取得、$a_LastName と $a_FirstName 配列に格納する
    SQL EXECUTE("SELECT LastName, FirstName FROM PERSONS WHERE LastName = ?";$a_LastName;$a_FirstName)
    If(Not(SQL End selection)) // レコードが検索されたら

       SQL LOAD RECORD(SQL All Records) // 全レコードをロードする
       SQL CANCEL LOAD

       $UseSQL:=True // データを統合する方法を選択

       If($UseSQL) // SQLリクエストを使用する場合
          SQL LOGOUT // リモートデータベースからログアウトする
          SQL LOGIN(SQL_INTERNAL;"user";"password") // ローカルデータベースにログイン
  // これ以降、すべてのSQL命令はローカルデータベース上で実行されます
  // $a_LastName と $a_FirstName 配列中のデータをローカルのPERSONSテーブルに保存
          SQL EXECUTE("INSERT INTO PERSONS(LastName, FirstName) VALUES (:$a_LastName, :$a_FirstName);")
          SQL CANCEL LOAD

       Else // 4Dコマンドを使用する場合

          REDUCE SELECTION([PERSONS];0)
          ARRAY TO SELECTION($a_LastName;[PERSONS]LastName;$a_FirstName;[PERSONS]FirstName)

       End if
    End if
    SQL LOGOUT // 接続を閉じる
 End if
```

システム変数およびセット

コマンドが正しく実行されると、OKシステム変数は1に、そうでなければ0に設定されます。

SQL EXECUTE SCRIPT

SQL EXECUTE SCRIPT (scriptPath ; errorAction {; attribName ; attribValue} {; attribName2 ; attribValue2 ; ... ; attribNameN ; attribValueN})

引数	型		説明
scriptPath	テキスト	⇒	実行するSQLスクリプトが書かれたファイルの完全パス名
errorAction	倍長整数	⇒	スクリプト実行中にエラーが発生した場合のアクション
attribName	テキスト	⇒	使用する属性の名前
attribValue	テキスト	⇒	属性の値

説明

SQL EXECUTE SCRIPTコマンドを使用して、scriptPathで指定されたスクリプトファイルに書かれた一連のSQLステートメントを実行できます。このコマンドはローカルマシン (ローカルの4Dまたは4D Server上のストアドプロシージャ) でのみ実行できます。またこのコマンドはカレントデータベース (内部あるいはエクスターナルデータベース) に対して動作します。

Note: このコマンドは直接あるいはODBC経由で開かれた外部接続では使用できません。

scriptPath引数には実行するSQL文が書かれたテキストファイルの完全パス名を渡します。パス名は現在のシステムのシンタックスを使用して表現されなければなりません。scriptPathに空の文字列 ("") を渡すと、標準のファイルを開くダイアログボックスが表示され、実行するファイルをユーザが選択できます。

Note: SQL EXPORT DATABASEとSQL EXPORT SELECTIONコマンドは自動でこのスクリプトファイルを作成します。

errorAction 引数を使用して、スクリプト実行中にエラーが発生した場合の動作を設定できます。""テーマ中の以下の3つの定数を使用できます:

定数	型	値	コメント
SQL On error abort	倍長整数	1	エラーイベント時、4Dは即座にスクリプト実行を停止します。
SQL On error confirm	倍長整数	2	エラーイベント時、4Dはエラーを説明するダイアログを表示し、スクリプト実行を停止するか続行するかをユーザーが選択できます。
SQL On error continue	倍長整数	3	エラーイベント時、4Dはそれを無視し、スクリプトの実行を続行します。

attribNameとattribValue引数は組で渡さなければなりません。これらの引数はスクリプト実行のための特定の属性として使用されることを意図しています。現在のバージョンの4D では、一つの属性をattribNameに渡せます。""テーマの以下の定数を使用できます:

定数	型	値	コメント
SQL Use Access Rights	文字列	SQL_Use_Access_Rights	スクリプトのSQLコマンドの実行中に適用されるアクセス権を制限するために使用します。この属性を使用する場合、attribValueには1または0を渡します: • attribValue = 1: 4Dはカレントユーザーのアクセス権を使用します。 • attribValue = 0 (または属性を指定しない場合): 4Dはアクセスを制限しません。デザイナー権限が使用されます。

(SET DATABASE PARAMETERのセレクター28や45で) 4Dログファイルが有効にされていると、SQLコマンドが実行されるごとに以下の情報が書き込まれます:

- SQLコマンドのタイプ
- コマンドの影響を受けるレコード数
- コマンドの実行時間
- エラーを検知するごとに:
  - エラーコード
  - 可能であればエラーテキスト

スクリプトが (エラーなく) 正しく実行されると、OK変数に1が設定されます。エラーが発生した場合、OKシステム変数に0が設定されるか、またはerrorAction引数に設定に基づき:

- errorAction がSQL On error abort (値1) の場合、OKは0に設定されます。
- errorActionがSQL On error confirm (値2) の場合、ユーザが処理を中断するとOKに0が、続行を選択すると1が設定されます。
- errorActionがSQL On error continue (値3) の場合、OKは常に1が設定されます。

Note: 大量のデータ読み込みなど、メモリを消費するアクションを実行するためにこのコマンドを使用する場合、一時的にSQLオプションを無効にするために新しいSQLのALTER DATABASEコマンドの実行を検討できます。

SQL EXPORT DATABASE

SQL EXPORT DATABASE (folderPath {; numFiles {; fileLimitSize {; fieldLimitSize}}})

引数	型		説明
folderPath	テキスト	→	書き出しフォルダーのパス名、または"" でフォルダー選択ダイアログボックスを表示
numFiles	倍長整数	→	フォルダーごとの最大ファイル数
fileLimitSize	倍長整数	→	書き出しファイルのサイズ制限値 (KB)
fieldLimitSize	倍長整数	→	この引数のサイズ以下のテキスト、Blob、ピクチャーフィールドの内容はメインのファイルに統合する (バイト単位)

説明

SQL EXPORT DATABASEコマンドはデータベースのすべてのテーブルのすべてのレコードをSQLフォーマットで書き出します。SQLではこのグローバルな書き出し処理は"ダンプ"と呼ばれます。

注: このコマンドは直接あるいはODBC経由で開かれた外部接続では使用できません。

テーブルごとに、コマンドは他のデータベースにデータを読み込む際に必要となるSQL文を含むテキストファイルを生成します。このファイルを直接SQL EXECUTE SCRIPTコマンドで使用して、他の4Dデータベースにデータを読み込むことができます。

書き出しファイルはfolderPathで指定された保存先フォルダ内に作成される"SQLExport"フォルダに配置されます。"SQLExport" フォルダが指定された場所に既に存在する場合、コマンドは警告なしにそれを置き換えます。
引数に空の文字列を渡すと、4Dは標準のフォルダを選択ダイアログボックスを表示します。デフォルトでダイアログボックスはセッションを開いたユーザのカレントフォルダーを表示します (Windows では"マイ ドキュメント"、Mac OS では"書類")。

書き出されるテーブルごとに、コマンドは以下のアクションを行います:

- 保存先フォルダにテーブル名によるサブフォルダを作成する
- サブフォルダ内にテキストファイル"Export.sql"を作成します。このファイルはBOM付きUTF-8でエンコードされます。ファイルには書き出されたデータに対応するSQLのINSERT 命令が含まれます。フィールドの値はコロンで区切られます。テーブル中のフィールド数よりデータが少ない場合がありますが、この場合残りのフィールドはNULLとして扱われます。
- テーブルにBLOBやピクチャ、またはテキストフィールド (レコードの外に格納されたテキスト) が含まれる場合、コマンドはデフォルトで追加のサブフォルダ"BLOBS"を"Export.sql" ファイルと同階層に作成し、必要なだけ"BlobsX"サブフォルダを作成します。これらのサブフォルダにはテーブルレコードのすべてのBLOB、ピクチャー、および外部テキストフィールドの内容が分離されたファイルとして格納されます。BLOB ファイルは"BlobXXXXX.BLOB"、テキストファイルは"TEXTXXXXX.TXT" (XXXXX はアプリケーションが生成するユニーク番号) という名称がつけられます。ピクチャーファイルはPICTXXXXX.ZZZZ (XXXXX はアプリケーションが生成するユニーク番号で、ZZZZ は拡張子)。可能であれば、ピクチャはオリジナルのネイティブフォーマットで書き出され、対応する拡張子が付けられます (.jpg, .png 等)。ネイティブフォーマットでの書き出しができない場合、ピクチャは4D の内部フォーマットで書き出され、.4PCT 拡張子が付けられます。
  このデフォルトの動作はオプションのfieldLimitSize引数を使用して、フィールドに含まれるデータのサイズに基づき調整することができます (後述)。
  注: リモートモードの4Dから**SQL EXPORT DATABASE**を実行した場合、この動作は異なります。このコンテキストでは、外部に保存されるデータは自動で"Export.sql"ファイルに含まれます。

numFiles 引数を渡すと、サブフォルダがnumFiles数以上のBLOB やピクチャを含まないよう、必要な応じて"BlobsX" サブフォルダを作成します。numFiles 引数が省略されると、デフォルトでコマンドはファイル数を200に制限します。0を渡すと、それぞれのサブフォルダは少なくとも1つのファイルを含みます。

fileLimitSize 引数を渡すと、ディスク上に作成されるそれぞれの"Export.sql"のサイズを (キロバイト単位で) 制限できます。作成された書き出しファイルのサイズがfileLimitSizeで設定した制限に達すると、4Dはレコードの書き込みを停止し、ファイルを閉じ、"ExportX.sql" (Xは一連番号を表す) という新しいファイルを同階層に作成します。このメカニズムにおいては、"ExportX.sql"ファイルの実際のサイズはfileLimitSizeを超える点に留意してください。なぜならサイズの制限を超えるときに書き出されているレコードが完全に書き出された後にファイルが閉じられるからです (つまり1レコードが2つのファイルに分割して書き出されることはないということです)。設定可能な最小サイズは100 KBで最大サイズ (デフォルト値) は100,000KB (100MB) です。

オプションのfieldLimitSize引数では、外部BLOB、ピクチャー、およびテキストフィールドデータがこの引数で設定されるサイズを下回る場合に、分離したファイルではなくメインの"Export.sql"に含めるそのサイズを設定します。この引数の目的はディスク上に作成されるサブフォルダーやファイルの数を制限することで書き出し処理を最適化することにあります。
この引数はバイト単位で設定します。例えば1000を渡すと、1000バイト以下の外部BLOB、ピクチャー、およびテキストフィールドはメインの書き出しファイルに組み込まれます。
書き出しファイルに組み込まれるバイナリーデータ (BLOBとピクチャー) はX'0f20'形式の16進フォーマットで書き込まれることに留意してください (標準のSQL16進記法、literal参照)。このフォーマットは4D SQLエンジンにより自動でサポートされます。
fieldLimitSize引数が省略されるとデフォルトで外部BLOB、ピクチャー、およびテキストフィールドの内容はサイズにかかわらず常に分離したファイルに書き出されます。

書き出しファイル中では、テーブル中のフィールドより値の数が少ない場合があります。この場合、空のフィールドはNULLとみなされます。フィールドにNULL値を渡すこともできます。

書き出しが正しく実行されるとOK変数に1が。そうでなければ0が設定されます。

SQL EXPORT SELECTION

SQL EXPORT SELECTION (aTable ; folderPath {; numFiles {; fileLimitSize {; fieldLimitSize}}})

引数	型		説明
aTable	テーブル	→	セレクションを書き出すテーブル
folderPath	テキスト	→	書き出しフォルダーのパス名、または"" でフォルダー選択ダイアログボックスを表示
numFiles	倍長整数	→	フォルダーごとの最大ファイル数
fileLimitSize	倍長整数	→	Export.sql ファイルの最大サイズ (KB)
fieldLimitSize	倍長整数	→	この引数のサイズ以下のテキスト、Blob、ピクチャーフィールドの内容はメインのファイルに統合する (バイト単位)

説明

SQL EXPORT SELECTIONコマンドは、aTable引数で指定した4Dテーブルのカレントセレクションを SQLフォーマットで書き出します。

このコマンドはSQL EXPORT DATABASEコマンドとほぼ同様のものですが、SQL EXPORT SELECTIONはaTable のカレントセレクションのみを書き出すのに対し、SQL EXPORT DATABASEはデータベース全体を書き出します。またSQL EXPORT DATABASEと異なり、SQL EXPORT SELECTIONは外部SQLデータベースでは動作しません。このコマンドはメインのデータベースでのみ使用できます。

これらのコマンドの動作と引数の説明についてはSQL EXPORT DATABASEコマンドを参照してください。

カレントセレクションが空の場合、コマンドはなにも行いません。この場合、保存先フォルダは空にされないことに留意してください。

書き出しが正しく実行されるとOK 変数に1 が、そうでなければ0 が設定されます。

SQL GET LAST ERROR

SQL GET LAST ERROR (errCode ; errText ; errODBC ; errSQLServer)

引数	型	説明
errCode	倍長整数	← エラーコード
errText	テキスト	← エラーテキスト
errODBC	テキスト	← ODBCエラーコード
errSQLServer	倍長整数	← SQLサーバネイティブエラーコード

説明

SQL GET LAST ERRORコマンドは、ODBCコマンドの実行中に発生した最後のエラーに関連する情報を返します。エラーの発生箇所としては、4Dアプリケーション、ネットワーク、ODBCソースなどが考えられます。

一般的に、このコマンドはON ERR CALLコマンドを用いて設定されたエラー処理用メソッド内で使用します。

- errCode引数にはエラーコードが返されます。
- errText引数にはエラーテキストが返されます。

残りの2つの引数には、エラーがODBCソースで生成された場合にのみ値が返されます。そうでない場合、空となります。

- errODBC引数にはODBCエラーコード (SQL state) が返されます。
- errSQLServer引数にはSQLサーバのネイティブエラーコードが返されます。

SQL GET OPTION

SQL GET OPTION (option ; value)

引数	型		説明
option	倍長整数	→	オプション番号
value	倍長整数, テキスト	←	オプション値

説明

SQL GET OPTIONコマンドは、optionに渡したオプションの現在のvalueを返します。

各種オプションとその関連する値についての詳細は、SQL SET OPTIONコマンドを参照してください。

システム変数およびセット

コマンドが正しく実行されるとOKシステム変数は1に、そうでなければ0に設定されます。

SQL LOAD RECORD

SQL LOAD RECORD {(numRecords)}

引数	型		説明
numRecords	倍長整数	→	ロードするレコード数

説明

SQL LOAD RECORDコマンドは、現在の接続において開かれたODBCソースからのレコードを1件以上4D内に取り込みます。

任意の引数numRecordsを使用し、取り出すレコード数を設定します:

- この引数を省略すると、コマンドはデータソースからカレントレコードを取り出します。この方法は、ループ中で一度に1レコードを受け取るデータ取得に対応します。
- numRecordsに整数値を渡すと、コマンドはnumRecords件のレコードを取り出します。
- SQL All Records定数（値 -1）を渡すと、コマンドはテーブルの全レコードを取り出します。

Note: 最後の2つの設定は、取得したデータが配列や4Dフィールドに関連付けられている場合にのみ意味を持ちます。

システム変数およびセット

コマンドが正しく実行されると、OKシステム変数は1に、そうでなければ0に設定されます。

SQL LOGIN

SQL LOGIN {(dataEntry ; userName ; password ; *)}

引数	型		説明
dataEntry	文字	⇒	外部データベース名、または 外部データソースのIPアドレス、または ODBCマネジャーのデータソース名、または ""で選択ダイアログボックスの表示
userName	文字	⇒	データソースに登録されているユーザー名
password	文字	⇒	データソースに登録されているパスワード
*	演算子	⇒	Begin SQL/End SQLへ適用される 省略した場合、適用しない(ローカルデータベース)、渡す場合、適用する

説明

SQL LOGINコマンドを使用すると、dataEntry引数で指定されたSQLデータソースへ接続することができます。このコマンドは、カレントプロセスでこのコマンドの後に実行される以下のSQLクエリの対象を指定します:

- SQL EXECUTEコマンド経由
- Begin SQL / End SQLタグ内に記述されているコード経由 (* 引数が渡された場合)

SQLデータソースは次のいずれかです:

- 直接アクセスする外部4D Serverデータベース
- 外部ODBCソース
- ローカル4Dデータベース (内部データベース)

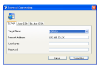
dataEntryには次の値のいずれかを渡します: IPアドレス、4Dデータベース公開名、ODBCデータソース名、空の文字列、またはSQL_INTERNAL定数。

- **IPアドレス**
  シンタックス: **IP:<IPAddress>{:<TCPPort>}{:ssl}**
  SQLクライアントは指定されたIPアドレスのコンピュータ上で実行される4D Serverデータベースに直接接続します。対象のコンピュータ上ではSQLサーバーが起動していなければなりません。TCPポートの数値を渡さない場合、デフォルトのポート (19812) が使用されます。SQLサーバー側でSQL公開ポートが変更されている場合、IPアドレスの後にポート番号を指定しなければなりません。SQLサーバー側では、データベース設定の "SQL" ページで、SQLサーバー用のTCPポート番号を変更できます。
  例4と5を参照してください。
  接続先のSQLサーバーでSSLが有効にされている場合 (データベース設定で設定可能)、サーバーで正しくリクエストを処理するために、IPアドレスの後にポート番号および":ssl"キーワードを追加しなければなりません (例題6参照)。

- **4Dデータベース公開名**
  シンタックス: **4D:<Publication_Name>**
  コマンドは、ネットワーク上で指定された名前に対応する公開名の4D Serverのデータベースに直接接続します。データベースのネットワーク上の公開名は、データベース設定の"Client-Server" ページで設定されています。
  例7を参照してください。
  Note: 接続する (4Dデータベースを公開する) 4D SQLサーバのTCPポート番号と、4DアプリケーションのSQLサーバ用のTCPポート番号は一致しなければなりません。

- **有効なODBCデータソース名**
  シンタックス: **ODBC:<My_DSN>** または **<My_DSN>**
  dataEntry引数に、ODBC Driverマネージャで設定したデータソースの名前を渡します。
  Note: 4Dの以前のバージョンとの互換性のため、接頭辞"ODBC:" を省略することも可能ですが、コードの読みやすさの理由から、接頭辞を使用することをお勧めします。例2を参照してください。

- **空の文字列**
  シンタックス: ""
  コマンドは接続ダイアログボックスを表示します。接続するデータソースは手動で入力できます。

  このダイアログボックスには複数のページがあります。TCP/IPページには次の要素があります。

  - **ターゲット名**: このメニューは2つのリストから構成されています。
    - 直接接続で最近開いたデータベースのリスト。このリストを更新するメカニズムは、4Dアプリケーションと同じです。ただし4DLinkファイルを格納するフォルダは"Favorites vXX"ではなく"Favorites SQL vXX" という名前です。
    - SQLサーバが開始されていて、そのSQL公開ポート番号が一致する4D Serverアプリケーションのリスト。このリストは、dataEntry引数を渡さずに**SQL LOGIN**コマンドを新たに呼び出す度に更新されます。データベースの名前の前に"^" 記号が付いている場合、安全なSSLモードを通して接続が行われたことを意味します。
  - **ネットワークアドレス**: このエリアはターゲット名で選択したデータベースのIPアドレスと、必要ならTCPポート番号を表示します。このエリアにIPアドレスを入力して接続ボタンを押して、対象の4D Serverデータベースに続することもできます。IPアドレスの後ろにコロン (:)を置いて、TCPポート番号を指定することも可能です。アドレスの後ろにコロン(:) を付けてポートの数値を入力します。例: 192.168.93.105:19855
  - **ユーザ名とパスワード**: これらのエリアを使用して、接続認証情報を入力します。
  - **ユーザDSNとシステムDSN** ページは、コンピュータのODBC Driverマネージャで指定されているシステムおよびユーザODBCデータソースのリストそれぞれ表示します。これらのページでデータソースを選択したり、認証情報を入力したりして、外部ODBCデータソースに接続できます。

  接続が確立されると、OKシステム変数が1に設定されます。そうでなければ0に設定され、エラーが生成されます。ON ERR CALLコマンドでインストールされているエラー処理メソッドでこのエラーをとらえることができます。

- **SQL_INTERNAL定数**
  シンタックス: SQL_INTERNAL
  コマンドは、続くSQLクエリを内部4Dデータベースに転送します。

警告: dataEntry 引数で使用される接頭辞 (IP, ODBC, 4D) は大文字でなければなりません。

userNameには、外部データソースへの接続が許可されたユーザの名前を渡します。例えば、Oracle(R)では、ユーザ名は"Scott"かもしれません。

passwordには、外部データソースへの接続が許可されたユーザのパスワードを渡します。例えば、 Oracle(R)では、パスワードは"tiger"かもしれません。

Note: 直接接続の場合、userNameとpassword引数に空の文字列を渡すと、4Dのパスワードシステムが対象データベースで有効になっていない場合のみ接続が受け入れられます。そうでなければ接続は拒否されます。

オプションの * 引数を使用して、Begin SQL / End SQL タグ内で実行されるSQLコードのターゲットを変更できます。この引数を渡さない場合、Begin SQL / End SQLタグ内に設定されているコードは、**SQL LOGIN**コマンドで指定されている設定を無視して、4Dの内部SQLエンジンに送られます。この引数を渡すと、Begin SQL / End SQLタグ内で実行されるSQLコードは、dataEntry 引数で指定されているソースへと送られます。

接続を終了してメモリを解放するには、SQL LOGOUTコマンドを実行するだけです。続くすべてのSQLクエリは、内部4D SQLデータベースへと送られます。
現在の接続を明示的に終了しないで**SQL LOGIN**を再び呼び出すと、接続は自動的に終了します。

注: SQL LOGINによる外部接続の試みに失敗した場合、内部4Dデータベースが自動でカレントデータソースとなります。

これらの引数はオプションです。引数をまったく渡さない場合、コマンドはODBCログインダイアログボックスを開き、外部データソースを選択することができます。

このコマンドのスコープはプロセスです。つまり、異なる2つの接続を行いたい場合、2つのプロセスを作成してそれぞれのプロセス内で各接続を実行しなくてはなりません。

例題 1

この文は、ODBCマネージャダイアログボックスを表示します:

```
SQL LOGIN
```

例題 2

ODBC プロトコル経由で外部データソース"MyOracle" と接続。SQL EXECUTEコマンドを用いて実行されたSQLクエリとBegin SQL / End SQLタグ内に収められているクエリは、この接続に送られます。

```
SQL LOGIN("ODBC:MyOracle";"Scott";"tiger";*)
```

例題 3

4Dの内部SQLカーネルに接続します:

```
SQL LOGIN(SQL_INTERNAL;$user;$password)
```

例題 4

デフォルトのTCPポートで接続を受け付ける、IPアドレス192.168.45.34のコンピュータで実行される4D Serverアプリケーションとの直接接続を開きます。SQL EXECUTEコマンド経由で実行されるSQLクエリはこの接続に送られます。Begin SQL / End SQLタグ内に収められているクエリはこの接続に送られません。

```
SQL LOGIN("IP:192.168.45.34";"John";"azerty")
```

例題 5

TCPポート20150で接続を待ち受ける、IPアドレス192.168.45.34のコンピュータで実行される4D Serverアプリケーションとの直接接続を開きます。SQL EXECUTEコマンド経由で実行されるSQLクエリとBegin SQL / End SQLタグ内に収められているクエリは、この接続に送られます。

```
SQL LOGIN("IP:192.168.45.34:20150";"John";"azerty";*)
```

例題 6

IPアドレス192.168.45.34マシン、およびデフォルトのTCPポートで動作する4D Serverアプリケーションに SSLで直接接続を開きます。4D ServerアプリケーションのSQLサーバー設定でSSLが有効にされています:

```
SQL LOGIN("IP:192.168.45.34:19812:ssl";"Admin";"ad156") // IPアドレスの後ろにポート番号と":ssl"が追加されていることに留意してください。
```

例題 7

ロー カルのネットワーク上で公開名"Accounts_DB"のデータベースを公開する4D Serverアプリケーションとの直接接続を開きます。両方のデータベースのSQLサーバ用のTCPポート (環境設定のSQL/設定ページで設定) は一致していなければなりません (デフォルトで19812)。SQL EXECUTEコマンド 経由で実行されるSQLクエリはこの接続に送られます。Begin SQL / End SQLタグ内に収められているクエリはこの接続に送られません。

```
SQL LOGIN("4D:Accounts_DB";"John";"azerty")
```

例題 8

SQL LOGINコマンドによる可能な接続を以下の例で表します。

```
ARRAY TEXT(aNames;0)
ARRAY LONGINT(aAges;0)
SQL LOGIN("ODBC:MyORACLE";"Marc";"azerty")
If(OK=1)
  ` 次のクエリは外部のOracleデータベースへ送られます。
   SQL EXECUTE("SELECT Name, Age FROM PERSONS";aNames;aAges)
  `次のクエリはローカルの4Dデータベースへ送られます。
   Begin SQL
      SELECT Name, Age
      FROM PERSONS
      INTO :aNames, :aAges;
   End SQL
  `次のSQL LOGINコマンドを使用して現在の外部ORACLEデータベースとの接続を終了します。
  `そして新たに外部MySQLデータベースと接続します。
   SQL LOGIN("ODBC:MySQL";"Jean";"qwerty";*)
   If(OK=1)
  `次のクエリは外部MySQLデータベースへ送られます。
      SQL EXECUTE("SELECT Name, Age FROM PERSONS";aNames;aAges)
  `次のクエリもまた外部MySQLデータベースへ送られます。
      Begin SQL
         SELECT Name, Age
         FROM PERSONS
         INTO :aNames, :aAges;
      End SQL
      SQL LOGOUT
  `次のクエリは4Dのローカルのデータベースへ送られます。
      Begin SQL
         SELECT Name, Age
         FROM PERSONS
         INTO :aNames, :aAges;
      End SQL
   End if
End if
```

システム変数およびセット

接続に成功すると、システム変数OKは1に、そうでなければ0に設定されます。

SQL LOGOUT

SQL LOGOUT
このコマンドは引数を必要としません

説明

SQL LOGOUTコマンドは、カレントプロセスの接続をクローズします（すでに接続されていれば）。接続が行われていない場合、コマンドは何も行いません。

システム変数およびセット

正しくログアウトがあ行われると、OKシステム変数は1に設定されます。そうでなければ0に設定されます。ON ERR CALLコマンドでインストールされたエラー処理メソッドを使用してこのエラーを処理できます。

SQL SET OPTION

SQL SET OPTION (option ; value)

引数	型		説明
option	倍長整数	→	設定するオプション番号
value	倍長整数, 文字	→	新しいオプションの値

説明

SQL SET OPTIONコマンドを使用して、optionに渡したオプションのvalueを変更します。

optionには"SQL"テーマにある次の定数のうちいずれかを渡します:

定数	型	値	コメント
SQL Asynchronous	倍長整数	1	0 = 同期接続 (デフォルト値) 1 (または非0値) = 非同期接続
SQL Charset	倍長整数	100	(SQLパススルー経由で) 外部ソースに送られるリクエストで使用されるテキストエンコーディング。変更はカレントプロセスのカレント接続に対して実行されます。 値: MIBEnum識別子 (Note2を参照)、または値 -2 (Note3を参照) デフォルト: 106 (UTF-8)
SQL Connection Timeout	倍長整数	5	SQL LOGINコマンド実行時にレスポンスを待ち受ける最大タイムアウト。この値が有効になるためには、接続を開く前に設定しなければなりません。 取りうる値: 秒数 デフォルト: タイムアウトしない
SQL Max Data Length	倍長整数	3	返されるデータの最大長
SQL Max Rows	倍長整数	2	結果グループの最大行数 (プレビューで使用)
SQL Query Timeout	倍長整数	4	SQL EXECUTEコマンドの実行時に応答を待機する最大タイムアウト 値: 秒数 デフォルト:タイムアウトしない

Notes:

1. 4Dの内部SQLカーネルを使用する場合、SQL Asynchronousオプションは意味を持ちません。この場合常に同期接続が使用されます。
2. MIBEnum番号は、次のアドレスで参照できます。 http://www.iana.org/assignments/character-sets
3. SQL Charsetのvalueとして-2を渡すと、4D SQLサーバが使用するエンコーディングは、自動で実行中のプラットフォームに合わせて適用されます (非Unicodeエンコーディング):
    - Windowsでは ISO8859-1が使用されます。
    - Mac OSではMAC-ROMANが使用されます。

システム変数およびセット

コマンドが正しく実行されるとOKシステム変数に1が、そうでなければ0が設定されます。

SQL SET PARAMETER

SQL SET PARAMETER (object ; paramType)

引数	型		説明
object	4Dオブジェクト	⇒	使用する4Dオブジェクト (変数、配列、フィールド)
paramType	倍長整数	⇒	引数タイプ

説明

SQL SET PARAMETERコマンドを使用すると、4D変数や配列、フィールドをSQLリクエストで使用することができます。

Note: リクエストテキスト内で、<< と >>記号の間に4Dオブジェクト（変数、配列、フィールド）を直接挿入できます（例1を参照）。詳細についてはSQLコマンドの概要を参照してください。

- objectの引数には、リクエストで使用する4Dのオブジェクト（変数、配列またはフィールド）を指定します。
- paramType引数には、パラメタのSQL型を渡します。値を渡すかSQLテーマにある次の定数のいずれかを使用することができます:

定数	型	値	コメント
SQL Param In	倍長整数	1	
SQL Param In Out	倍長整数	2	SQLストアードプロシージャーのコンテキストでのみ利用可能です (ストアードプロシージャー内で定義されるin-outパラメーター)
SQL Param Out	倍長整数	4	SQLストアードプロシージャーのコンテキストでのみ利用可能です (ストアードプロシージャー内で定義されるoutパラメーター)

SQLリクエスト内におかれた?記号が4Dオブジェクトの値で置き換えられます（標準のシンタックス）。
リクエストに複数の?記号が含まれる場合、**SQL SET PARAMETER**コマンドを複数回呼び出す必要があります。4Dオブジェクトの値は、コマンドの実行順に合わせてリクエスト内で順次割り当てられます。

警告: このコマンドはSQLリクエストに渡されるパラメーターを扱うために使用します。4DオブジェクトをSQLリクエストの結果に関連付けるためにSQL Param Outを使用することはできません。SQLリクエストの結果は例えばSQL EXECUTEコマンドの boundObj パラメーターを使用して取り出されます (SQLコマンドの概要参照)。**SQL SET PARAMETER**コマンドは主にリクエストに渡されるパラメーター (SQL Param In) を設定することを意図しています。SQL Param Out と SQL Param In Out はパラメーターを返すかもしれないSQLストアードプロシージャーのコンテキストで使用されるために予約されています。

例題 1

この例は、SQLリクエスト内に4D変数を直接記述しています:

```
C_TEXT(MyText)
C_LONGINT(MyLongint)

SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (<<MyText>>, <<MyLongint>>)"
For(vCounter;1;10)
   MyText:="Text"+String(vCounter)
   MyLongint:=vCounter
   SQL EXECUTE(SQLStmt)
   SQL CANCEL LOAD
End for
SQL LOGOUT
```

例題 2

上記と同じ例題をSQL SET PARAMETERコマンドを使用して書き直しています:

```
C_TEXT(MyText)
C_LONGINT(MyLongint)

SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (?,?)"
For(vCounter;1;10)
   MyText:="Text"+String(vCounter)
   MyLongint:=vCounter
   SQL SET PARAMETER(MyText;SQL Param In)
   SQL SET PARAMETER(MyLongint;SQL Param In)
   SQL EXECUTE(SQLStmt)
   SQL CANCEL LOAD
End for
SQL LOGOUT
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が、そうでなければ0が設定されます。

START SQL SERVER

START SQL SERVER
このコマンドは引数を必要としません

説明

START SQL SERVERコマンドを使用して、実行中の4Dのアプリケーションで、統合されたSQLサーバを起動させます。起動すると、SQLサーバは外部SQLクエリに応答します。

Note: このコマンドは4Dの内部SQLカーネルの機能には影響しません。内部クエリの際、SQLカーネルはいつでも利用できます。

システム変数およびセット

SQLサーバが正しく起動されるとOKシステム変数は1に、そうでなければ0に設定されます。

STOP SQL SERVER

STOP SQL SERVER
このコマンドは引数を必要としません

説明

STOP SQL SERVERコマンドは、実行済み4Dアプリケーションの統合SQLサーバを停止します。

SQLサーバが起動していた場合、すべてのSQL接続が中断します。そして以降、サーバは外部SQLクエリを受け入れません。SQLサーバが起動していなかった場合、このコマンドは何も行いません。

Note: このコマンドは4Dの内部SQLカーネルの機能には影響しません。内部クエリの際、SQLカーネルはいつでも利用できます。

USE EXTERNAL DATABASE

USE EXTERNAL DATABASE (sourceName {; user ; password})

引数	型	説明
sourceName	文字 →	接続するODBCデータソースの名前
user	文字 →	ユーザ名
password	文字 →	ユーザパスワード

互換性に関するメモ

このコマンドは4Dのバージョン11.3以降、SQL LOGINコマンドに置き換えられています。このコマンドは互換性のために保持されていますが、今後のバージョンでは保守されません。

システム変数およびセット

コマンドの実行が成功した場合、OKシステム変数は1で設定されます。それ以外の場合は0で設定され、エラーが発生します。

USE INTERNAL DATABASE

USE INTERNAL DATABASE
このコマンドは引数を必要としません

互換性に関するメモ

このコマンドは4Dのバージョン11.3以降、SQL LOGOUTコマンドに置き換えられています。このコマンドは互換性のために保持されていますが、今後のバージョンでは保守されません。

システム変数およびセット

コマンドの実行が成功した場合、OKシステム変数は1で設定されます。それ以外の場合は0で設定され、エラーが発生します。

Styled Text

"**Styled Text**" テーマには、マルチスタイルのテキストエリア(またの名を *rich text areas* とも言います)を管理・変更するためのコマンドや関数が用意されています。

リッチテキストを宣言するためには、プロパティリストの中の「マルチスタイル」にチェックを入れる必要があることに注意して下さい(*Design Reference* マニュアルの マルチスタイル (リッチテキストエリア) を参照して下さい)。"" テーマの OBJECT Is styled text を使用してテキストエリアのマルチスタイルがモードが有効になっているかどうかを調べることができます。

汎用コマンドとマルチスタイルエリアの関係性

サポートされているタグ

ST COMPUTE EXPRESSIONS New 14.0

ST FREEZE EXPRESSIONS New 14.0

ST GET ATTRIBUTES

ST Get content type New 14.0

ST Get expression New 14.0

ST GET OPTIONS New 14.0

ST Get plain text Updated 14.0

ST Get text

ST GET URL New 14.0

ST INSERT EXPRESSION New 14.0

ST INSERT URL New 14.0

ST SET ATTRIBUTES

ST SET OPTIONS New 14.0

ST SET PLAIN TEXT

ST SET TEXT

汎用コマンドとマルチスタイルエリアの関係性

テキストオブジェクト管理コマンド

プログラムでテキストオブジェクトを操作するために使用することのできるコマンドは、テキストに統合されたスタイルタグを無視します。以下のコマンドが関連します:

- ユーザーインターフェース テーマ
  HIGHLIGHT TEXT
  GET HIGHLIGHT

文字列を操作するコマンドとともにこれらのコマンドを使用する際、新しいコマンドST Get plain textを使用してフォーマット文字をフィルターしなければならないことに留意してください:

```
HIGHLIGHT TEXT([Products]Notes;1;Length(OBJECT Get plain text([Products]Notes))+1)
```

- オブジェクト(フォーム) テーマ
  オブジェクトのスタイルを変更するために使用するコマンド (例えばOBJECT SET FONT) は選択された文字列ではなく、指定されたオブジェクト全体を対象とします。コマンドが実行されたときオブジェクトにフォーカスがない場合、変更はオブジェクト (テキストエリア) とそれに割り当てられた変数に同時に適用されます。オブジェクトにフォーカスがある場合、変更はそのオブ ジェクトに適用され、割り当てられた変数には適用されません。この場合変更は変数に対してオブジェクトがフォーカスを失ったときにのみ適用されます。テキストエリアに対するプログラムを行う際はこの原則を忘れないでください。

"デフォルトスタイルタグを格納"がそのオブジェクトに対して選択されている場合、これらのコマンドを使用すると、オブジェクトに保存されているタグが更新されます。

汎用コマンドとマルチスタイルエリアの関係性

4D v14 以降、OBJECT SET RGB COLORS や OBJECT SET FONT STYLE といった汎用コマンドとマルチスタイルエリアとの関係性が新しくなりました。

以前のバージョンの4Dでは、これらのコマンドのどれかを実行すると、エリア内に挿入されたカスタムのスタイルタグを全て変更してしまいました。v14からは、デフォルトのプロパティ(とデフォルトのタグで保存されたプロパティ)のみが変更されるようになりました。カスタムのスタイルタグはそのままの状態を維持します。

例えば、以下の様なマルチスタイルエリアにデフォルトのタグが保存されていた場合を考えます。

This is the word red

このエリアのプレーンテキストは以下のようになります:

```
<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#009900">This is the word <span
style="color:#D81E05">red</span></span>
```

以下のコードを実行した場合:

```
OBJECT SET COLOR(*;"myArea";-(Blue+(256*Yellow)))
```

4D v14 では、赤文字の部分はそのまま赤文字として残ります。

4D v14	以前のバージョン
This is the word red	This is the word red
`<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#0000FF">This is the word <span style="color:#D81E05">red</span></span>`	`<span style="font-family:'Segoe UI';font-size:9pt;text-align:left;font-weight:normal;font-style:normal;text-decoration:none;color:#0000FF;"><span style="background-color:#FFFFFF">This is the word red</span></span>`

これが適用される汎用コマンドは、以下の5つです:

OBJECT SET RGB COLORS
OBJECT SET COLOR
OBJECT SET FONT
OBJECT SET FONT STYLE
OBJECT SET FONT SIZE

マルチスタイルエリアにおいては、汎用コマンドはデフォルトのスタイルを設定するためだけに使用されるべきです。データベースの実行中にスタイルを管理するためには、"Styled Text" テーマ内のコマンドを使用することが推奨されます。

Get edited text コマンド

Get edited text (フォームイベントテーマ) がリッチテキストエリアで使用されると、コマンドはすべてのスタイルタグを含む現在のエリアのすべてのテキストを返します。

編集された生テキスト (タグなしのテキスト) を取り出すには、ST Get plain textコマンドを使用しなければなりません:

```
OBJECT Get plain text(Get edited text)
```

クエリおよび並び替えコマンド

マルチスタイルオブジェクトに対して行われるクエリや並び替えはオブジェクトに保存されたスタイルタグを考慮に入れます。単語中でスタイルの変更が行われた場合、その単語の検索は失敗します。

有効な検索や並び替えを行うには、ST Get plain textコマン ドを使用します。例えば:

```
QUERY BY FORMULA([MyTable];OBJECT Get plain text([MyTable]MyFieldStyle)="very well")
```

行末の自動標準化

データベースで扱われるテキストがより多くのプラットフォームで互換性があることを保証するために、v14以降、4Dは自動的に行末を標準化し、単一文字('\r' (キャリッジリターン))分のスペースを確保するようにしました。この標準化は、マルチスタイルテキストまたは標準テキストを内包しているフォームオブジェクト(変数またはフィールド)まで適用されます。ネイティブでない行末、または複数の文字の組み合わせ ('\r\n' 等) は単一の文字'\r' として認識されます。

XML 標準(マルチスタイルテキストフォーマット)と適合するため、マルチスタイルテキストコマンドも、オブジェクトに関連付けられていないテキスト変数の行末を標準化することに注意して下さい。

この原理により、マルチスタイルテキストコマンドや、HIGHLIGHT TEXTのようなコマンドをマルチプラットフォームのコンテキストで簡単に使用できるようになります。しかしながら、異なるソースからのテキストを扱う処理をする場合には、このことを考慮に入れなければなりません。

サポートされているタグ

ダイナミックタグ

4D のマルチスタイルテキストエリアでは以下のタグを使用することができます。

4D式

このタグは、テキストに4D式(式、メソッド、フィールド、変数、コマンド、等)を挿入します。この式は以下のタイミングで評価されます:

- 式が挿入されたとき
- オブジェクトがロードされたとき
- ST COMPUTE EXPRESSIONS コマンドが実行されたとき
- ST FREEZE EXPRESSIONS コマンドに第二引数 * が渡された状態でが実行されたとき

式の評価された値は タグに保存されません。参照のみが保存されます。

URL

Visible label

このタグはテキストにURLを挿入します。
例:

```
<span><a href="http://www.4d.com/">4D Web Site</a></span>
```

ユーザーリンク

Click here

"ユーザーリンク"は見た目はURLと同じですが、クリックしてもソースは自動的には開きません。参照としてはどんな文字列を渡す事もできますが、クリックされたときに起こるアクションはデベロッパによってプログラムされた内容によります。
これはつまりタグがクリックされときに、URLではなく、ファイル、4Dメソッド等を開いたり実行したりすることができるということです。ST Get content typeコマンドによって、ユーザーリンクがクリックされたかどうかを検知することもできます。

ユーザーリンクは ST SET TEXT コマンドを使用して定義されています。以下の様に定義します:

```
 ST SET TEXT(txtVar;"This is a user link: <span style=\"-d4-ref-user:'UserLink'\">User Label</span>";$start;$end)
```

カスタムのタグ

v14以降、スタイル付テキストにどんなタグも挿入することが出来るようになりました。例えば<img src="http://doc.4d.com/pictures/ja.png">のようなタグでもです。タグは標準テキストのコードに保存されますが、解釈されたり表示されたりすることはありません。これは例えば、HTMLフォーマットのEメールの中に画像を挿入するような場合に特に有効です。

スタイルタグ

ここでは <SPAN> タグの属性としてサポートされる4Dのリッチテキストエリアのスタイルを説明します。これらの属性を使用してカスタムスタイルを指定することも可能です。ここで説明されている属性のみを4Dはサポートします。

フォント名

<SPAN STYLE="font-family: フォントファミリ"> ... </SPAN>

フォントサイズ

<SPAN STYLE="font-size: 20pt"> ... </SPAN>

フォントスタイル

- **ボールド**
  <SPAN STYLE="font-weight: bold"> ... </SPAN>
- **イタリックまたは普通**
  <SPAN STYLE="font-style: italic"> ... </SPAN>
  <SPAN STYLE="font-style: normal"> ... </SPAN>
- **下線**
  <SPAN STYLE="text-decoration: underline"> ... </SPAN>
- **取り消し線**
  <SPAN STYLE="text-decoration:line-through">...</SPAN>
  注：Mac OS では取り消し線がサポートされていません。指定しても表示されませんが、プログラムで管理することはできます。

フォントカラー

<SPAN STYLE="color:green"> ... </SPAN>
または
<SPAN STYLE="color:#006CCC">...</SPAN>

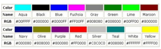
背景色 (Windowsのみ)

<SPAN STYLE="background-color:green"> ... </SPAN>
または
<SPAN STYLE="background-color:#006CCC">...</SPAN>

注: Mac OSではこの属性が無視されます。オブジェクトが更新されるときにこの属性は取り除かれます。

カラーの値

フォントカラーと背景色属性では、カラー値としてRGBの16進およびCSSで定義されたHTMLカラー名を指定できます。:



Color	■	■	■	■	■	■	■	■
Name	Aqua	Black	Blue	Fuchsia	Gray	Green	Lime	Maroon
RGB	#00FFFF	#000000	#0000FF	#FF00FF	#808080	#008000	#00FF00	#800000

Color	■	■	■	■	■	■	□	■
Name	Navy	Olive	Purple	Red	Silver	Teal	White	Yellow
RGB	#000080	#808000	#800080	#FF0000	#C0C0C0	#008080	#FFFFFF	#FFFF00

ST COMPUTE EXPRESSIONS

ST COMPUTE EXPRESSIONS ({* ;} object {; startSel {; endSel}})

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
startSel	倍長整数	→	選択範囲の開始地点
endSel	倍長整数	→	選択範囲の終了地点

説明

ST COMPUTE EXPRESSIONSコマンドは、object 引数で指定したテキストフィールドまたはテキスト変数のダイナミック 4D 式を更新します。

マルチスタイルテキストエリアで使用されている 4D 式の詳細に関しては、ST INSERT EXPRESSION コマンドの詳細を参照して下さい。

このコマンドは object で指定したオブジェクト内の4D 式の結果を、カレントの内容に応じて更新してそれを表示します。例えば、挿入された4D 式が時刻であった場合、表示される時刻は **ST COMPUTE EXPRESSIONS** コマンドを使用するたびに変更されます。4D 式は以下のときにも更新されます:

- 挿入されたとき
- オブジェクトがロードされたとき
- ST FREEZE EXPRESSIONS コマンドにおいて、2番目の * を渡して4D式が固定化されたとき

ST COMPUTE EXPRESSIONS コマンドは SPAN タグも含めてスタイル付テキストを変更しません。object で指定されたオブジェクト内に表示された標準テキストのみ変更します。処理された値はスタイル付テキストの中には保存されず、参照のみが保存されます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

指定されたobject はフォーカスになっている必要はありません。しかしながら、オブジェクトはフォームに含まれている必要があります。そうでない場合には**ST COMPUTE EXPRESSIONS** コマンドは何もしません。

任意の startSel 引数と endSel 引数はオブジェクト内のテキストの選択範囲を指定します。startSel と endSel は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- startSel と endSel の両方を渡した場合、**ST COMPUTE EXPRESSIONS** コマンドは指定された範囲内の4D式のみ変更します。
- startSel のみを指定した場合、もしくは endSel の値がオブジェクト内の総文字数より大きい場合は、startSel の位置からテキストの終わりまでの全ての4D式が変更されます。
- startSel と endSel の両方を省略した場合、指定されたオブジェクト内の全ての4D式は変更されます。

4D では、選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります:

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

(*) これら二つの定数を使用する際には、object 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

注: もし startSel 引数が endSel 引数より大きい場合、コマンドは何も行わず、OK変数は0に設定されます。(ただし endSel が0である場合を除く)

例題

テキストの選択範囲に含まれる参照を更新する場合を考えます:

```
ST COMPUTE EXPRESSIONS(*;"myText";ST Start highlight;ST End highlight)
```

ST FREEZE EXPRESSIONS

ST FREEZE EXPRESSIONS ({* ;} object {; startSel {; endSel}}{; *})

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
startSel	倍長整数	→	選択範囲の開始地点
endSel	倍長整数	→	選択範囲の終了地点
*	演算子	→	渡した場合、固定化する前に4D式を更新

説明

ST FREEZE EXPRESSIONSコマンドは、object 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内の4D 式の内容を固定化します。これにより、ダイナミックな4D式はスタティックなテキストへと変換され、object に関連付けられた参照は解除されます。

マルチスタイルテキストエリアの中で使用される4D 式の詳細については、ST INSERT EXPRESSION を参照して下さい。

ST FREEZE EXPRESSIONS コマンドは、変更された4D式の値を保存します。この操作は特に、object をマルチスタイルエリア外で使用する(エクスポート、ディスクファイルへの保存、印刷等)際に必要になります。なぜなら、エリアのそのものには、4D式への参照しか保存されていないからです。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

任意の startSel 引数と endSel 引数はobject 内のテキストの選択範囲を指定します。startSel と endSel は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- startSel と endSel の両方を渡した場合、**ST FREEZE EXPRESSIONS** コマンドは指定された範囲内の4D式のみ固定化します。
- startSel のみを指定した場合、もしくは endSel の値がobject 内の総文字数より大きい場合は、startSel の位置からテキストの終わりまでの全ての4D式が固定化されます。
- startSel と endSel の両方を省略した場合、指定されたobject 内の全ての4D式が固定化されます。

4D では、選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります:

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

(*) これら二つの定数を使用する際には、object 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

注: もし startSel 引数が endSel 引数より大きい場合、コマンドは何も行わず、*OK* 変数は0に設定されます。(ただし endSel が0である場合を除く)

デフォルトでは、4D式は固定化される直前に再評価されるわけではありません。4D式を評価しなおして固定化したい場合には、2 番目の* 演算子を渡します。

例題

テキストの冒頭にカレントの時刻を挿入し、レコードに保存する前に固定化する場合を考えます:

```
  //テキストの冒頭に時刻を挿入
 ST INSERT EXPRESSION(*;StyledText_t;"Current time";1)
  //式を固定
 ST FREEZE EXPRESSIONS(*;"StyledText_t";1)
```

ST GET ATTRIBUTES

ST GET ATTRIBUTES ({* ;} object ; startSel ; endSel ; attribName ; attribValue {; attribName2 ; attribValue2 ; ... ; attribNameN ; attribValueN})

引数	型		説明
*	演算子	⇒	指定時: objectはオブジェクト名 (文字) 省略時: objectは変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または テキストフィールドまたは変数 (* 省略時)
startSel	倍長整数	⇒	テキスト選択の開始位置
endSel	倍長整数	⇒	テキスト選択の終了位置
attribName	倍長整数	⇒	取得する属性
attribValue	変数	⇐	属性の現在の値

説明

ST GET ATTRIBUTES コマンドは、objectで指定したフォームオブジェクト中で選択されたテキストのスタイル属性を取得するために使用します。

オプションの * 引数を渡した場合、object引数にはオブジェクト名 (文字列) を渡します。コマンド実行時にオブジェクトにフォーカスがある場合、コマンドは編集中のオブジェクトに関する情報を返します。他方オブジェクトにフォーカスがない場合、コマンドはオブジェクトのデータソース (フィールドや変数) に関する情報を返します。
* 引数を省略した場合、object 引数にはフィールドや変数を指定します。この場合文字列ではなくフィールドまたは変数への参照を渡します。コマンドはこのフィールドや変数に関する情報を返します。

startSelとendSel引数を使用して、オブジェクト中でスタイル属性を取得するテキストを選択します。startSelには選択する最初の文字位置を、endSelには選択する文字の最後の位置に1加えた値を渡します。endSelに0をセットすることで、自動的にテキストの最後を指定(startSelに1をセットすることでテキストの最初を指定)できます。startSelとendSelの値が等しい場合や、startSelがendSelよりも大きい場合(endSelが0の場合を除く・上記参照)、エラーが返されます。
startSelとendSel、エリア中に既に存在するスタイルタグを考慮に入れません。文字数のカウントは (テキストからスタイルタグを取り除いた) 生テキストを基に行います。

4D では選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡せる規定済み定数を用意しています。これらの定数は "Multistyle Text" テーマ内にあります:

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

(*) これらの定数を使用するためには、object にオブジェクト名を渡す必要があります。フィールドまたは変数への参照を渡した場合、コマンドはオブジェクトの全てのテキストに対して適用されます。

attribName 引数に取得する属性の名前を、attribValue引数には属性値を受け取る変数を渡します。attribName 引数を指定するにはMultistyle Text Attributesテーマの定数の一つを使用しなければなりません。

定数	型	値	コメント
Attribute background color	倍長整数	8	*attValue* = (Windowsのみ) 16進値またはHTMLカラー名
Attribute bold style	倍長整数	1	*attValue* = 0: 選択部からボールド属性を取り除きます *attValue* = 1: 選択部にボールド属性を適用します
Attribute font name	倍長整数	5	*attValue* = フォントファミリー名 (文字)
Attribute italic style	倍長整数	2	*attValue* = 0: 選択部からイタリック属性を取り除きます *attValue* = 1: 選択部にイタリック属性を適用します
Attribute strikethrough style	倍長整数	3	*attValue* = 0: 選択部から取り消し線属性を取り除きます *attValue* = 1: 選択部に取り消し線属性を適用します
Attribute text color	倍長整数	7	*attValue* = 16進値またはHTMLカラー名
Attribute text size	倍長整数	6	*attValue* = ポイント数 (数値)
Attribute underline style	倍長整数	4	*attValue* = 0: 選択部から下線属性を取り除きます *attValue* = 1: 選択部に下線属性を適用します

必要なだけ属性/値の組を渡すことができます。

attribName 属性の値が選択された文字列中全体で同じ場合、attribValueにそれが返されます。値が異なる場合やobject がSPANタグを含まない場合、以下の値が返されます:

attribName	属性値が一致しない場合やSPANタグが含まれない場合のattValue
Attribute background color	FFFFFFFF
Attribute bold style	2
Attribute font name	"" (空の文字列)
Attribute italic style	2
Attribute strikethrough style	2
Attribute text color	FFFFFFFF
Attribute text size	-1
Attribute underline style	2

例題

フォームに[Table_1]というStyledTextフィールドが表示されています。このオブジェクトはマルチスタイルで"StyledText_t"という名前がついているとします。選択されたテキストと、太字スタイル属性についてのステータスを取得したいという場合、オブジェクト名かフィールド参照のどちらかを使用して処理することができます。

- オブジェクト名を使用する方法:

```
$text:=ST Get text(*;"StyledText_t";ST Start highlight;ST End highlight)
ST GET ATTRIBUTES(*;"StyledText_t";ST Start highlight;ST End highlight;Attribute bold style;$bold)
```

- フィールド名を使用する方法:

```
GET HIGHLIGHT([Table_1]StyledText;$Begin_1;$End_1)
$text:=ST Get text([Table_1]StyledText;$Begin_1;$End_1)
ST GET ATTRIBUTES([Table_1]StyledText;$Begin_1;$End_1;Attribute bold style;$bold)
```

システム変数およびセット

このコマンド実行後、エラーがなければOKシステム変数に1が設定されます。そうでなければ0が設定されます。これは特にスタイルタグが正しく評価できなかった場合に発生します (正しくない、あるいは失われたタグ)。

エラーの場合、変数は変更されません。テキストが評価される際に変数にエラーが発生すると、4Dはテキストをプレーンテキストに変換します。結果、"<"、">"、"&"文字はHTML実体参照に変換されます。

ST Get content type

ST Get content type ({* ;} object {; startSel {; endSel {; startBlock {; endBlock}}}}) -> 戻り値

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
startSel	倍長整数	→	選択範囲の開始地点
endSel	倍長整数	→	選択範囲の終了地点
startBlock	倍長整数	←	選択範囲内の、同一のタイプの開始地点
endBlock	倍長整数	←	選択範囲内の、同一のタイプの終了地点
戻り値	倍長整数	↩	内容のタイプ

説明

ST Get content typeコマンドは、object 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内で見つかったコンテンツの型を返します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。このとき、オブジェクトがフォーカスされていた場合にはコマンドは編集中のテキストに関しての情報を返し、オブジェクトがフォーカスされていない場合にはコマンドはオブジェクトのデータソースの情報を返します。
省略時には object 引数でフィールドまたは変数を指定します。この場合にはコマンドは変数またはフィールドの情報を返します。

省略時には object 引数でフィールドまたは変数を指定します。この場合にはコマンドは変数またはフィールドの情報を返します。

任意の startSel 引数と endSel 引数はobject 内のテキストの選択範囲を指定します。startSel と endSel は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- startSel と endSel の両方を渡した場合、**ST Get content type** コマンドは指定された範囲内に限りコンテンツを評価します。
- startSel のみを指定した場合、もしくは endSel の値がobject 内の総文字数より大きい場合は、コマンドは startSel の位置からテキストの終わりまでの範囲内のコンテンツのみ評価されます。
- startSel と endSel の両方を省略した場合、現在選択されている範囲のコンテンツのみが評価されます。

4D では、選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります:

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

(*) これら二つの定数を使用する際には、object 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

注: もし startSel 引数が endSel 引数より大きい場合、コマンドは何も行わず、*OK* 変数は0に設定されます(ただし endSel が0である場合を除く)。

任意の startBlock 引数と endBlock 引数は、オブジェクト内、もしくはオブジェクトの中で選択された範囲の中で、タイプが連続する位置を探し、その最初と最後の文字を返します。例えば、選択範囲内に4D式に続いて標準テキストが含まれていた場合、startBlock とendBlock はそれぞれ4D式の開始地点と終了地点を返します。この操作は選択範囲内の全てを処理するためにループさせることができます。

コマンドは、選択範囲内を調べた結果特定できたタイプを示す値を返します。返される値は、"Multistyle Text" テーマ内にある、以下のどれかになります:

定数	型	値	コメント
ST Expression type	倍長整数	2	セレクションには式参照のみ含まれます。
ST Mixed type	倍長整数	3	セレクションには少なくとも二つの異なる型のコンテンツが含まれます。
ST Plain type	倍長整数	0	セレクションにはテキストのみ含まれ、参照はありません。
ST Unknown tag type	倍長整数	4	セレクションには、未知の型のタグのみ含まれます。
ST Url type	倍長整数	1	セレクションにはURL参照のみ含まれます。
ST User type	倍長整数	5	セレクションには、カスタムの参照のみ含まれます。

例題

エリア内で選択されたコンテンツのタイプによって表示されるコンテキストメニューを変えたい場合:

```
Case of
   :(Form event=On Clicked)
  //選択された範囲を取得
      GET HIGHLIGHT(*;"myText";startSel;endSel)
      If(Contextual click & (Macintosh control down=False)) //コンテキストメニューを呼び出し
         If(startSel=endSel) // コンテンツが選択されていない場合
  //一部のコマンドのみ有効化
            DISABLE MENU ITEM(<>menu_STYLEDTEXT;2)
            DISABLE MENU ITEM(<>menu_STYLEDTEXT;4)
            ENABLE MENU ITEM(<>menu_STYLEDTEXT;6)
            ...
         Else // コンテンツの型を取得
            CT_Texttype:=ST Get content type(*;"myText";startSel;endSel)
            Case of // 異なるタイプによって処理を変える
               :(CT_Texttype=ST Url type)
                  DISABLE MENU ITEM(<>menu_STYLEDTEXT;6)
                  ENABLE MENU ITEM(<>menu_STYLEDTEXT;7)
                  ...
               :(CT_Texttype=ST Expression type)
                  DISABLE MENU ITEM(<>menu_STYLEDTEXT;6)
                  DISABLE MENU ITEM(<>menu_STYLEDTEXT;7)
                  ...
               Else
                  ENABLE MENU ITEM(<>menu_STYLEDTEXT;6)
                  DISABLE MENU ITEM(<>menu_STYLEDTEXT;7)
                  ...
            End case
         End if
         GET MOUSE($xCoord;$yCoord;$ButtonState)
         $AlphaVar:=Dynamic pop up menu(<>menu_STYLEDTEXT;"";$xCoord;$yCoord)
         startSel:=-3
         endSel:=-3
      End if
      ...
   End if
```

ST Get expression

ST Get expression ({* ;} object {; startSel {; endSel}}) -> 戻り値

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
startSel	倍長整数	→	選択範囲の開始地点
endSel	倍長整数	→	選択範囲の終了地点
戻り値	テキスト	←	4D式の内容

説明

ST Get expressionコマンドは、object 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内のカレントセレクションの中で、最初に見つけた 4D 式を返します。

このコマンドは、オブジェクト内に挿入された4D式の内容を返します。(結果は、例えば "mymethod" や"[table1]field1" 等になります) 4D式の値は返されません。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。このとき、オブジェクトがフォーカスされていた場合にはコマンドは編集中のテキストに関しての情報を返し、オブジェクトがフォーカスされていない場合にはコマンドはオブジェクトのデータソースの情報を返します。
省略時には object 引数でフィールドまたは変数を指定します。この場合にはコマンドは変数またはフィールドの情報を返します。

任意の startSel 引数と endSel 引数はobject 内のテキストの選択範囲を指定します。startSel と endSel は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- startSel と endSel の両方を渡した場合、**ST Get expression** コマンドは指定された範囲内に限り4D式を探します。
- startSel のみを指定した場合、もしくは endSel の値がオブジェクト内の総文字数より大きい場合は、コマンドは startSel の位置からテキストの終わりまでの範囲内の4D式を探します。
- startSel と endSel の両方を省略した場合、指定されたオブジェクトが編集中であれば、そのとき選択されている範囲のテキスト内を探します。

4D では、選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります:

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

(*) これら二つの定数を使用する際には、object 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

注: もし startSel 引数が endSel 引数より大きい場合、コマンドは何も行わず、*OK* 変数は0に設定されます(ただし endSel が0である場合を除く)。

選択範囲の中に4D式が何もない場合、コマンドは空の文字列を返します。

例題 1

ダブルクリックイベントがあると、4D式が実際にあるかどうかをチェックし、あった場合にはユーザーがそれを変更できるようにその値を取得したダイアログボックスを表示する、という場合:

```
 Case of
    :(Form event=On Double Clicked)
       GET HIGHLIGHT(*;"StyledText_t";startSel;endSel)
       If(ST Get content type(*;"StyledText_t";startSel;endSel)=ST Expression type)
          vExpression:=ST Get expression(*;"StyledText_t";startSel;endSel)
          $winRef:=Open form window("Dial_InsertExpr";Movable form dialog box;Horizontally Centered;Vertically
Centered;*)
          DIALOG("Dial_InsertExpr")
          If(OK=1)
             ST INSERT EXPRESSION(*;"StyledText_t";vExpression;startSel;endSel)
             HIGHLIGHT TEXT(*;"StyledText_t";startSel;endSel)
          End if
       End if
 End case
```

例題 2

ユーザーがリンクをクリックしたときに4Dメソッドを実行したい場合:

```
 Case of
    :(Form event=On Clicked)
  //セレクションを取得
       HIGHLIGHT TEXT(*;"myText";startSel;endSel)
       If(startSel#endSel) //選択されたコンテンツが存在
  //コンテンツの型を取得
          $CT_type:=ST Get content type(*;"myText";startSel;endSel)
          If($CT_type=ST User type) //これはユーザーリンク
             MyMethod //4Dメソッドを実行
          End if
       End if
 End case
```

ST GET OPTIONS

ST GET OPTIONS ({* ;} object ; option ; value {; option2 ; value2 ; ... ; optionN ; valueN})

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
option	倍長整数	⇒	取得したいオプション
value	倍長整数	⇐	オプションのカレントの値

説明

ST GET OPTIONSコマンドは、object 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内で動作中のオプションのカレントの値を取得します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。このとき、オブジェクトがフォーカスされていた場合にはコマンドは編集中のテキストに関しての情報を返し、オブジェクトがフォーカスされていない場合にはコマンドはオブジェクトのデータソースの情報を返します。
省略時には object 引数でフィールドまたは変数を指定します。この場合にはコマンドは変数またはフィールドの情報を返します。

option 引数には、取得したいオプションを指定するコードを渡します。コマンドはそのオプションのカレントの値を value 引数に返します。 option 引数には "Multistyle Text" テーマ内にある、以下の定数を使用することができます:

定数	型	値	コメント
ST Expressions display mode	倍長整数	1	value 引数には以下のどちらかが入ります: • ST Values: 式の計算された値を表示します(デフォルト) • ST References: 式のソースの文字列を表示します
ST References	倍長整数	1	式のソースの文字列を表示します。
ST Values	倍長整数	0	4D式の、計算された値を表示します。

ST Get plain text

ST Get plain text ({* ;} object {; refMode}) -> 戻り値

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字) 省略時: objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または テキストフィールドまたは変数 (* 省略時)
refMode	倍長整数	→	Mode for handling references found in the text
戻り値	テキスト	↩	タグなしのテキスト

説明

ST Get plain text コマンドは*とobject引数で指定したテキスト変数やフィールドからスタイルタグを取り除き、プレーンテキストを返します。

オプションの * 引数を渡した場合、object引数にはオブジェクト名 (文字列) を渡します。コマンド実行時にオブジェクトにフォーカスがある場合、コマンドは編集中のオブジェクトに関する情報を返します。他方オブジェクトにフォーカスがない場合、コマンドはオブジェクトのデータソース (フィールドや変数) に関する情報を返します。
* 引数を省略した場合、object 引数にはフィールドや変数を指定します。この場合文字列ではなくフィールドまたは変数への参照を渡します。コマンドはこのフィールドや変数に関する情報を返します。

任意の refMode 引数は、object 内で見つかった参照の返し方を指定します。refMode 引数には、"Multistyle Text" テーマ内にある以下の定数のどれかを渡して下さい(定数は一つ、または複数組み合わせて渡す事ができます):

定数	型	値	コメント
ST 4D Expressions as sources	倍長整数	2	4D式参照のオリジナルの文字列が返されます。
ST 4D Expressions as values	倍長整数	1	4D 式参照は評価された形で返されます(フォーム内のデフォルトの動作)。
ST References as spaces	倍長整数	0	それぞれの参照は、ノンブレークスペース文字として返されます(他のコマンドで使用されるデフォルトの動作)
ST Tags as plain text	倍長整数	64	タグのラベルは標準テキストとして返されます。例えば、'<img src="test.jpg" alt="picture">my picture</img>'というタグの標準テキストは "my picture" となります(フォームでのデフォルトの動作)。
ST Tags as XML code	倍長整数	128	XMLのコードは標準テキストとして返されます。例えば、'<img src="test.jpg" alt="picture">my picture</img>'というタグの標準テキストは '<img src="test.jpg" alt="picture">my picture</img>' となります。
ST Text displayed with 4D Expression sources	倍長整数	86	テキストは、4D式のオリジナルの文字列とともに、表示されたままの状態が返されます。 既定済みの定数の組み合わせの 2+4+16+64 に対応します。
ST Text displayed with 4D Expression values	倍長整数	85	テキストは、4D式が評価された形で、フォームで表示されたままの形で返されます。 既定済みの定数の組み合わせの 1+4+16+64 に対応します。
ST URL as labels	倍長整数	4	「こちらのサイトまでどうぞ」といったようなURLの表示ラベルが返されます(フォームのデフォルトの機能)
ST URL as links	倍長整数	8	"http://www.4d.com" のようにリンクが返されます。
ST User links as labels	倍長整数	16	ユーザーリンクの表示ラベルが返されます(フォームのデフォルトの機能)
ST User links as links	倍長整数	32	ユーザーリンクの中身が返されます。

Note: 標準テキストに関しては、refMode 引数にどの値を渡しても(どのモードで扱っても)変化はありません。言い換えると、refMode を使って違いが現れるのはテキストに参照が含まれていた場合のみです。

例題 1

マルチスタイル付きのテキストフィールドからテキスト"very nice"を探します。値は以下のような形式で保存されています: "The weather is very nice **today**".

```
QUERY BY FORMULA([Comments];ST Get plain text([Comments]Weather)="@very nice@")
```

Note: このコンテキストでは、スタイルタグがテキストに含まれるため、以下のコードでは期待通りの結果が得られません:

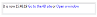
```
QUERY([Comments];[Comments]Weather="@very nice@")
```

例題 2

"MyArea" というマルチスタイルエリア内に以下の様なテキストが入っている場合:

```
<span>It is now <span style="-d4-ref:'Current time:C178'"> </span> <a href="http://www.4d.com">Go to the 4D site</a> or <span style="-d4-ref-user:'openW'">Open a window</span></span>
```

このテキストは以下の様に表示されます:



It is now 15:48:19 Go to the 4D site or Open a window

以下は、あるコードを実行した場合とそれぞれに対して返ってくる値です:

```
 $txt :=ST Get plain text(*;"myArea";ST References as spaces)
  // $txt = "It is now    or " (spaces)
 $txt :=ST Get plain text(*;"myArea";ST 4D Expressions as values)
  // $txt = "It is now 15:48:19  or  "
 $txt :=ST Get plain text(*;"myArea";ST 4D Expressions as sources)
  // $txt = "It is now Current time  or  "
 $txt :=ST Get plain text(*;"myArea";ST URL as links)
  //$txt = "It is now   http://www.4d.com or  "
 $txt :=ST Get plain text(*;"myArea";ST Text displayed with 4D Expression values)
  //$txt = "It is now 15:48:19 Go to the 4D site or Open a window"
 $txt :=ST Get plain text(*;"myArea";ST Text displayed with 4D Expression sources)
  //$txt = "It is now Current time Go to 4D site or Open a window"
 $txt :=ST Get plain text(*;"myArea";ST User links as labels)
  //$txt = "It is now    or Open a window"
 $txt :=ST Get plain text(*;"myArea";ST User links as links)
  //$txt = "It is now    or openW"
```

システム変数およびセット

このコマンド実行後、エラーがなければOKシステム変数に1が設定されます。そうでなければ0が設定されます。これは特にスタイルタグが正しく評価できなかった場合に発生します (正しくない、あるいは失われたタグ)。

エラーの場合、変数は変更されません。テキストが評価される際に変数にエラーが発生すると、4Dはテキストをプレーンテキストに変換します。結果、"<"、">"、"&"文字はHTML実体参照に変換されます。

ST Get text

ST Get text ({* ;} object {; startSel {; endSel}}) -> 戻り値

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時 objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時)または、テキストフィールドか変数 (* 省略時)
startSel	倍長整数	→	文字選択の開始位置
endSel	倍長整数	→	文字選択の終了位置
戻り値	テキスト	↩	スタイルタグを含むテキスト

説明

ST Get text コマンドは object 引数で指定されたフィールドや変数中のスタイル付きテキストを返します。

オプションの * 引数を渡した場合、object引数にはオブジェクト名 (文字列) を渡します。コマンド実行時にオブジェクトにフォーカスがある場合、コマンドは編集中のオブジェクトに関する情報を返します。他方オブジェクトにフォーカスがない場合、コマンドはオブジェクトのデータソース (フィールドや変数) に関する情報を返します。
* 引数を省略した場合、object 引数にはフィールドや変数を指定します。この場合文字列ではなくフィールドまたは変数への参照を渡します。コマンドはこのフィールドや変数に関する情報を返します。

コマンドはテキストに割り当てられたスタイルタグとともにテキストを返します。これは例えばスタイルを保持したままテキストのコピーとペーストを行う場合に使用します。

オプションのstartSelとendSel引数を使用して、object中のテキストを選択できます。startSelとendSelの値はプレーンテキストの選択に使用され、スタイルタグは無視されます。

- startSel と endSelを省略すると、ST Get text はobjectに含まれるすべてのテキストを返します。
- startSel と endSelを渡すと、ST Get text はこれらの引数により選択されたテキストを返します。

4D では選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡せる規定済み定数を用意しています。これらの定数は "Multistyle Text" テーマ内にあります:

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

(*) これらの定数を使用するためには、object にオブジェクト名を渡す必要があります。フィールドまたは変数への参照を渡した場合、コマンドはオブジェクトの全てのテキストに対して適用されます。

startSel と endSel の値が等しい場合、またはstartSel が endSelよりも大きい場合、エラーが返されます。

システム変数およびセット

このコマンド実行後、エラーがなければOKシステム変数に1が設定されます。そうでなければ0が設定されます。これは特にスタイルタグが正しく評価できなかった場合に発生します (正しくない、あるいは失われたタグ)。

エラーの場合、変数は変更されません。テキストが評価される際に変数にエラーが発生すると、4Dはテキストをプレーンテキストに変換します。結果、"<"、">"、"&"文字はHTML実体参照に変換されます。

ST GET URL

ST GET URL ({* ;} object ; urlText ; urlAddress {; startSel {; endSel}})

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
urlText	テキスト	⇐	リンクの表示テキスト
urlAddress	テキスト	⇐	URL アドレス
startSel	倍長整数	⇒	選択範囲の開始地点
endSel	倍長整数	⇒	選択範囲の終了地点

説明

ST GET URLコマンドは、object 引数で指定されたスタイル付テキストフィールドまたはテキスト変数内で見つかった最初のURL の表示テキストとアドレスを返します。

表示テキストとURL アドレスは、それぞれ urlText 引数と urlAddress 引数に返されます。もし選択範囲内にURL が含まれない場合は、これらの引数には空の文字列が返されます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。このとき、オブジェクトがフォーカスされていた場合にはコマンドは編集中のテキストに関しての情報を返し、オブジェクトがフォーカスされていない場合にはコマンドはオブジェクトのデータソースの情報を返します。
省略時には object 引数でフィールドまたは変数を指定します。この場合にはコマンドは変数またはフィールドの情報を返します。

任意の startSel 引数と endSel 引数はオブジェクト内のテキストの選択範囲を指定します。startSel と endSel は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- startSel と endSel の両方を渡した場合、**ST GET URL** コマンドは指定された範囲内に限り URL を探します。
- startSel のみを指定した場合、もしくは endSel の値がobject 内の総文字数より大きい場合は、コマンドはstartSel の位置からテキストの終わりまでの範囲内に限り URL を探します。
- startSel と endSel の両方を省略した場合、現在選択中の範囲内に限りURL を探します。

4D では、選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります:

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

(*) これら二つの定数を使用する際には、object 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

注: もし startSel 引数が endSel 引数より大きい場合、コマンドは何も行わず、*OK* 変数は0に設定されます(ただし endSel が0である場合を除く)。

例題

ダブルクリックイベントがあると、URLが実際にあるかどうかをチェックし、あった場合にはユーザーがそれを変更できるようにその値を取得したダイアログボックスを表示する、という場合について考えます:

```
 Case of
    :(Form event=On Double Clicked)
       GET HIGHLIGHT(*;"StyledText_t";startSel;endSel)
       If(ST Get content type(*;"StyledText_t";startSel;endSel)=ST Url type) //URL
          ST GET URL(*;"StyledText_t";vTitle;vURL;startSel;endSel)
          $winRef:=Open form window("Dial_InsertURL";Movable form dialog box;Horizontally Centered;Vertically
Centered;*)
          SET WINDOW TITLE("URL settings")
          DIALOG("Dial_InsertURL")
          If(OK=1)
             ST INSERT URL(*;"StyledText_t";vTitle;vURL;startSel;endSel)
             HIGHLIGHT TEXT(*;"StyledText_t";startSel;startSel+1)
          End if
       End if
 End case
```

ST INSERT EXPRESSION

ST INSERT EXPRESSION ({* ;} object ; expression {; startSel {; endSel}})

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列) 省略時: object は変数またはフィールド
object	Object	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
expression	テキスト	→	挿入したい4D式と(任意の)フォーマット
startSel	倍長整数	→	選択範囲の開始地点
endSel	倍長整数	→	選択範囲の終了地点

説明

ST INSERT EXPRESSIONコマンドは、object 引数で指定したスタイル付テキストフィールドまたはテキスト変数内に、4D 式への参照を挿入します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

expression 引数には、オブジェクト内にて評価したい4D式を渡します。有効な4D式とは、値を返す文字列です。expression 引数には、フィールド、変数、4Dコマンド、値を返す宣言、プロジェクトメソッド等を指定することができます。

4D式は、引用符("")で囲まれている必要があります。

注: expression 引数にピクチャ型の変数を渡すことはできません。

expression で指定した4D式から返ってきた値がキャリッジリターンとタブを含んでいた場合、4D式が入っているオブジェクトに合わせてそのテキストを表示します。例えばこのとき、キャリッジリターンは改行として扱われます。

expression 引数に、任意のフォーマット情報を渡すことによって、4D式のフォーマット形式を指定することが出来ます。この場合、以下の書式で指定する必要があります:

```
"String(value;format)"
```

value 引数には4D式そのものを渡し、format 引数には適用したい書式を渡します。format には以下のどれかを指定することができます:

- 数字に対しては、(存在するかしないかに関わらず)いかなる表示形式も指定することができます。例えば、"###,##" のような形です。
- 日付に対しては、存在する日付のフォーマットを指定する数字を渡すことができます。このとき、"Date Display Formats" テーマ内にある定数(System date short 等)を使用することができます。
- 時刻に対しては、存在する時刻のフォーマットを指定する数字を渡すことができます。このとき、"Time Display Formats" テーマ内にある定数(System time short 等)を使用することができます。

例えば、以下の様な形になります:

```
 "String([Table_1]Field_1;System date short)"
```

特に何も指定しなければ、expression の **値** がマルチスタイルテキストエリアに表示されます。また、ST SET OPTIONS コマンドを使用することによって強制的に**参照**を表示させることもできます。

任意の startSel 引数と endSel 引数はobject 内のテキストの選択範囲を指定します。startSel と endSel は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- startSel と endSel の両方を渡した場合、**ST INSERT EXPRESSION** コマンドは指定された範囲内のテキストを expression からの結果で置換します。
- startSel のみを指定した場合、もしくは endSel の値がobject 内の総文字数より大きい場合は、startSel の位置からテキストの終わりまでの全てのテキストが expression からの結果で置換されます。
- startSel と endSel の両方を省略した場合、指定されたobject が編集中であれば、そのとき選択されている範囲のテキストが4D式からの結果で置換されるか、範囲が選択されていなければカーソルのある位置に挿入されます。

4D では、選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります:

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

(*) これら二つの定数を使用する際には、object 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

注: もし startSel 引数が endSel 引数より大きい場合、コマンドは何も行わず、OK変数は0に設定されます(ただし endSel が0である場合を除く)。

例題

選択したテキストをプロジェクトメソッドの結果で置き換える場合を考えます:

```
 ST INSERT EXPRESSION(*;"myText";"MyMethod";ST Start highlight;ST End highlight)
```

ST INSERT URL

ST INSERT URL ({* ;} object ; urlText ; urlAddress {; startSel {; endSel}})

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
urlText	テキスト	→	リンクの表示テキスト
urlAddress	テキスト	→	URL アドレス
startSel	倍長整数	→	選択範囲の開始地点
endSel	倍長整数	→	選択範囲の終了地点

説明

ST INSERT URLコマンドは、object 引数で指定したスタイル付テキストフィールドまたはテキスト変数内に、URL のリンクを挿入します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

urlText 引数には、オブジェクト内で表示される、リンクの表示テキストを渡します。例えば、"4D Web Site" や "Follow this link for more information" などのテキストラベルです。"http://www.4d.com" のような、アドレスそのものを指定することもできます。

urlAddress 引数には、"http://www.4D.com" のように、ブラウザに表示させたいWebサイトの完全なアドレスを渡します。

任意の startSel 引数と endSel 引数はobject 内のテキストの選択範囲を指定します。startSel と endSel は標準テキストのみをカウントし、スタイルタグは文字数としてはカウントされません。参照は一文字としてカウントされるということに注意して下さい。

- startSel と endSel の両方を渡した場合、**ST INSERT URL** コマンドは指定された範囲内のテキストのみ urlText で指定したテキストと置換します。
- startSel のみを指定した場合、もしくは endSel の値がオブジェクト内の総文字数より大きい場合は、startSel の位置からテキストの終わりまでの全ての文字が urlText で置換されます。
- startSel と endSel の両方を省略した場合、指定されたobject が編集中であればそのとき選択されている範囲のテキストが urlText で置換されるか、範囲が選択されていなければカーソルのある位置に4D式が挿入されます。

4D では、選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡す定数をご用意しています。これらの定数は "Multistyle Text" テーマ内にあります:

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

(*) これら二つの定数を使用する際には、object 引数にオブジェクト名を渡してあげる必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

Note: もし startSel 引数が endSel 引数より大きい場合、コマンドは何も行わず、*OK* 変数は0に設定されます(ただし endSel が0である場合を除く)。

リンクは挿入されれば既に有効になっています。ラベルを、Windows では **Ctrl+クリック**、Mac OS X では**Command+クリック** することによって既定のブラウザでurlAddress で指定したページを開くことができます。

例題

オブジェクト内で選択されたテキストを、4D のウェブサイトへのリンクで置き換えたい場合を考えます:

```
vTitle:="4D Web Site"
vURL:="http://www.4d.com/"
ST INSERT URL(*;"myText";vTitle;vURL;ST Start highlight;ST End highlight)
```

ST SET ATTRIBUTES

ST SET ATTRIBUTES ({* ;} object ; startSel ; endSel ; attribName ; attribValue {; attribName2 ; attribValue2 ; ... ; attribNameN ; attribValueN})

引数	型		説明
*	演算子	⇒	指定時: objectはオブジェクト名 (文字) 省略時: objectは変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または フィールドまたは変数 (* 省略時)
startSel	倍長整数	⇒	新しいテキスト選択の開始位置
endSel	倍長整数	⇒	新しいテキスト選択の終了位置
attribName	文字	⇒	設定する属性
attribValue	文字, 倍長整数	⇒	新しい属性値

説明

ST SET ATTRIBUTES コマンドを使用して、objectで指定したフォームオブジェクト中の1つ以上のスタイル属性を変更できます。

オプションの * 引数を渡した場合、object引数にはオブジェクト名 (文字列) を渡します。コマンド実行時にオブジェクトにフォーカスがある場合、コマンドは編集中のオブジェクトにのみ適用され、(フィールドや変数などの) データソースには適用されません。変更がソース (およびこの同じソースを使用する他のオブジェクト) に転送されるのは、フォーカスが外れるか**Enter**キーが押されて、オブジェクトへの編集が有効化された場合のみです。オブジェクトにフォーカスがない場合、コマンドは直接データソースに適用され、変更は即座に同じソースを使用する他のオブジェクトに転送されます。
* 引数を省略した場合、object 引数にはフィールドや変数を指定します。この場合文字列ではなくフィールドまたは変数への参照を渡します。コマンドは直接データソースに適用され、変更は即座に同じソースを使用する他のオブジェクトに転送されます。

Note: テキスト型のフィールドにのみスタイル属性を使用できます。文字型のフィールドは事前に設定された長さがあるため、スタイルを追加すると、データが失われることがあります。

属性の定義はHTMLスタイルのタグをテキストに挿入したり変更したりすることにより行われます (この点に関する詳細は*Design Reference*を参照してください)。**ST SET ATTRIBUTES**は、objectがマルチスタイルプロパティを設定されていないテキストのフォームオブジェクトを指定している場合でも、すべてのケースでスタイルタグを挿入することに留意してください。

startSelとendSel引数はobject内でスタイルの変更を適用するテキストを選択するために使用します。startSelにはスタイルの変更を行う最初の文字の位置を、endSelに変更を行う最後の文字の位置に1加えた数を渡します (最後の位置の文字は変更の対象となりません)。endSelに0をセットすることで、自動的にテキストの最後を指定(startSelに1をセットすることでテキストの最初を指定)できます。
endSelがobject中の文字数より大きい場合、startSelからテキストの最後までが変更されます。
startSelがendSelより大きい場合(endSelが0の場合を除く・上記参照)、コマンドはなにも行わず、OKシステム変数が0に設定されます。
startSelとendSelの値はエリアに既に存在するスタイルタグを考慮しません。これらの引数は生のテキスト (スタイルタグがフィルタされたテキスト) をもとに評価されます。

4D では選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡せる規定済み定数を用意しています。これらの定数は "Multistyle Text" テーマ内にあります:

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

(*) これらの定数を使用するためには、object にオブジェクト名を渡す必要があります。フィールドまたは変数への参照を渡した場合、コマンドはオブジェクトの全てのテキストに対して適用されます。

attribName とattribValueには変更する属性に対応する名前と値を渡します。必要なだけ属性/値の組を渡すことができます。attribName 引数を指定するためにはMultistyle Text Attributesテーマの定義済み定数を使用します。attribValue引数に渡す値は、attribName 引数に基づきます:

定数	型	値	コメント
Attribute background color	倍長整数	8	*attValue* = (Windowsのみ) 16進値またはHTMLカラー名
Attribute bold style	倍長整数	1	*attValue* = 0: 選択部からボールド属性を取り除きます *attValue* = 1: 選択部にボールド属性を適用します
Attribute font name	倍長整数	5	*attValue* = フォントファミリー名 (文字)
Attribute italic style	倍長整数	2	*attValue* = 0: 選択部からイタリック属性を取り除きます *attValue* = 1: 選択部にイタリック属性を適用します
Attribute strikethrough style	倍長整数	3	*attValue* = 0: 選択部から取り消し線属性を取り除きます *attValue* = 1: 選択部に取り消し線属性を適用します
Attribute text color	倍長整数	7	*attValue* = 16進値またはHTMLカラー名
Attribute text size	倍長整数	6	*attValue* = ポイント数 (数値)
Attribute underline style	倍長整数	4	*attValue* = 0: 選択部から下線属性を取り除きます *attValue* = 1: 選択部に下線属性を適用します

カラー
attribName 引数に Attribute text color や Attribute background color 定数を渡した場合、attribValueにはHTMLカラー名か16進のカラー値を文字で渡さなければなりません:

HTMLカラー名	16進値
Aqua	#00FFFF
Black	#000000
Blue	#0000FF
Fuchsia	#FF00FF
Gray	#808080
Green	#008000
Lime	#00FF00
Maroon	#800000
Navy	#000080
Olive	#808000
Purple	#800080
Red	#FF0000
Silver	#C0C0C0
Teal	#008080
White	#FFFFFF
Yellow	#FFFF00

例題

この例題ではテキストのサイズやカラーのほか、ボールドおよび下線属性を2番目から4番目の文字に設定します:

```
 ST SET ATTRIBUTES([MyTable]MyField;2;5;Attribute font name;"Arial";Attribute text size;10;Attribute underline style;1;Attribute bold style;1;Attribute text color;"Blue")
```

システム変数およびセット

このコマンド実行後、エラーがなければOKシステム変数に1が設定されます。そうでなければ0が設定されます。これは特にスタイルタグが正しく評価できなかった場合に発生します (正しくない、あるいは失われたタグ)。

エラーの場合、変数は変更されません。テキストが評価される際に変数にエラーが発生すると、4Dはテキストをプレーンテキストに変換します。結果、"<"、">"、"&"文字はHTML実体参照に変換されます。

ST SET OPTIONS

ST SET OPTIONS ({* ;} object ; option ; value {; option2 ; value2 ; ... ; optionN ; valueN})

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
option	倍長整数	→	設定したいオプション
value	倍長整数	→	オプションの新しい値

説明

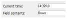
ST SET OPTIONSコマンドは、object 引数で指定されたスタイル付テキストフィールドまたはテキスト変数に関する様々なオプションを変更することができます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

修正したいオプションを指定する値を option 引数に、新しく設定したい値を value に渡します。このどちらの引数にも、"Multistyle Text" テーマ内にある以下の定数を使用することができます:

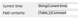
定数	型	値	コメント
ST Expressions display mode	倍長整数	1	value 引数には以下のどちらかが入ります: • ST Values: 式の計算された値を表示します(デフォルト) • ST References: 式のソースの文字列を表示します
ST References	倍長整数	1	式のソースの文字列を表示します。
ST Values	倍長整数	0	4D式の、計算された値を表示します。

値の表示:



Current time: 14:39:10
Field contents: Bravo

ソースの表示:



Current time: String(Current time)
Field contents: [Table_1]Comment

例題

以下のコードは、エリアの表示モードを切り替えます:

```
ST GET OPTIONS(*;"StyledText_t";ST Expressions display mode;$exprValue)
If($exprValue=1)
   ST SET OPTIONS(*;"StyledText_t";ST Expressions display mode;ST Values)
Else
   ST SET OPTIONS(*;"StyledText_t";ST Expressions display mode;ST References)
End if
```

ST SET PLAIN TEXT

ST SET PLAIN TEXT ({* ;} object ; newText {; startSel {; endSel}})

引数	型		説明
*	演算子	➡	指定時、objectはオブジェクト名 (文字列)。省略時、オブジェクトは変数またはフィールド
object	フォームオブジェクト	➡	オブジェクト名 (*指定時)、または変数/フィールド (*省略時)
newText	テキスト	➡	挿入するテキスト
startSel	倍長整数	➡	選択の開始位置
endSel	倍長整数	➡	選択の終了位置

説明

ST SET PLAIN TEXTコマンドはobject引数で指定されたマルチスタイルテキストや変数に、newText引数に渡されたテキストを挿入します。このコマンドはobject引数のプレーンテキストにのみ適用され、そこに含まれるスタイルタグを変更しません。

ST SET TEXTコマンドと異なり、**ST SET PLAIN TEXT**はプレーンテキストのみを挿入します。newTextにスタイルタグを含めることはできません。もし"<"や">"、"&"文字が含まれる場合、これらは標準の文字として扱われ、HTML実体参照に変換されます:

- '&' -> &amp;
- '<' -> &lt;
- '>' -> &gt;

オプションの * 引数を渡した場合、object引数にはオブジェクト名 (文字列) を渡します。コマンド実行時にオブジェクトにフォーカスがある場合、コマンドは編集中のオブジェクトにのみ適用され、(フィールドや変数などの) データソースには適用されません。変更がソース (およびこの同じソースを使用する他のオブジェクト) に転送されるのは、フォーカスが外れるかEnterキーが押されて、オブジェクトへの編集が有効化された場合のみです。オブジェクトにフォーカスがない場合、コマンドは直接データソースに適用され、変更は即座に同じソースを使用する他のオブジェクトに転送されます。
* 引数を省略した場合、object 引数にはフィールドや変数を指定します。この場合文字列ではなくフィールドまたは変数への参照を渡します。コマンドは直接データソースに適用され、変更は即座に同じソースを使用する他のオブジェクトに転送されます。

newTextには挿入するプレーンテキストを渡します。

オプションのstartSelとendSel引数を使用すると、object中で選択するテキストを指定できます。startSelとendSelの値はプレーンテキストを対象として、スタイルタグは考慮されません。コマンドの動作はオプションのstartSelとendSel引数の値により変化します:

- startSelとendSelを省略すると、**ST SET PLAIN TEXT**はobjectのすべてのテキストをnewTextで置き換えます。
- startSelのみを渡すか、startSelとendSelの値が同じ場合、**ST SET PLAIN TEXT**はobjectのstartSelの位置にnewTextを挿入します。
- startSelとendSel両方を渡すと、**ST SET PLAIN TEXT**はこれらの引数で指定された範囲のプレーンテキストをnewTextで置き換えます。
- endSelに0をセットすることで、自動的にテキストの最後を指定(startSelに1をセットすることでテキストの最初を指定)できます。

4D では、選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡す定数を提供しています。これらの定数は "Multistyle Text" テーマ内にあります。

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

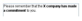
(*) この定数を使用する際には、object 引数にオブジェクト名を渡す必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

置き換えられる最初の文字のスタイルがnewTextのテキスト全体で使用されます。

startSelがendSelよりも大きい場合(endSelが0の場合を除く・上記参照)、テキストは変更されず、OK変数に0が設定されます。

例題

リッチテキスト (マルチスタイル) 設定された以下のようなフォーム上の変数があります:

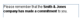


Please remember that the **X company has made a commitment** to you.

ここに、テキストフィールドに格納されている会社名を挿入したいとします。この名前には例えば"&"のような文字が含まれているかもしれません。この場合**ST SET PLAIN TEXT**コマンドを使用する必要があります:

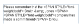
```
ST SET PLAIN TEXT(myStyledText;[Company]Name;33;34)
```

以下のような結果になります:



Please remember that the **Smith & Jones company has made a commitment** to you.

変数に格納されているプレーンテキストは以下のようになります:



Please remember that the <SPAN STYLE="font-weight:bold"> Smith &amp; Jones </SPAN> <SPAN STYLE="font-weight:bold">company has made a commitment</SPAN> to you.

挿入されたテキストは追加のスタイルタグ中にあることが分かります。このスタイルタグは挿入前の文字に設定されていたスタイルタグに対応します。このメカニズムにより、すべてのケースでリッチテキストエリアが正しく表示されるようになります。

注: このケースで ST SET TEXT コマンドを使用すると、4Dはテキストを挿入できない場合があります。エンコードされていない"&"のような文字が存在す る場合、変数中のスタイルタグの解釈が妨げられるためです。詳細はこのコマンドの説明を参照してください。

システム変数およびセット

このコマンド実行後、エラーが発生しなければOKシステム変数が1に、そうでなければ0に設定されます。エラーは特にスタイルタグが正しく評価できなかった場合に発生します (タグが正しくないあるいはタグが足りない)。

エラーが発生した場合、変数は変更されません。テキストが評価されるときに変数上でエラーが発生すると、4Dはテキストをプレーンテキストに変換します。結果"<"や">"、"&"文字はHTML実体参照に変換されます。

ST SET TEXT

ST SET TEXT ({* ;} object ; newText {; startSel {; endSel}})

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字) 省略時: objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または フィールドまたは変数 (* 省略時)
newText	テキスト	→	挿入するテキスト
startSel	倍長整数	→	テキスト選択開始位置
endSel	倍長整数	→	テキスト選択終了位置

説明

ST SET TEXT コマンドはobject引数で指定されたスタイル付きのフィールドや変数に、newText引数で渡されたテキストを挿入します。このコマンドはobject引数のプレーンテキストにのみ適用され、含まれるスタイルタグは更新しません。このコマンドはスクリーンに表示されているスタイル付きテキストをプログラムで変更するために使用できます。

オプションの * 引数を渡した場合、object引数にはオブジェクト名 (文字列) を渡します。コマンド実行時にオブジェクトにフォーカスがある場合、コマンドは編集中のオブジェクトにのみ適用され、(フィールドや変数などの) データソースには適用されません。変更がソース (およびこの同じソースを使用する他のオブジェクト) に転送されるのは、フォーカスが外れるか**Enter**キーが押されて、オブジェクトへの編集が有効化された場合のみです。オブジェクトにフォーカスがない場合、コマンドは直接データソースに適用され、変更は即座に同じソースを使用する他のオブジェクトに転送されます。
* 引数を省略した場合、object 引数にはフィールドや変数を指定します。この場合文字列ではなくフィールドまたは変数への参照を渡します。コマンドは直接データソースに適用され、変更は即座に同じソースを使用する他のオブジェクトに転送されます。

newTextには挿入するテキストを渡します。**ST SET TEXT**コマンドは<SPAN>型のタグを含むリッチ (マルチスタイル) テキスト挿入するために使用します。他のすべてのケース、特に<や>、&を含むプレーンテキストの場合、ST SET PLAIN TEXT コマンドを使用しなければなりません。**ST SET TEXT**コマンドに < や >、& を含むプレーンテキストを渡すと、コマンドは何も行いません。これは"a>b"のようなテキストをエンコードしないでリッチテキストに挿入すると、内部的な<SPAN>タグの解析が妨げられるからです。この場合">"文字は"&gt;"にエンコードされなければな りません。これはST SET PLAIN TEXTにより自動で行われます (このコマンドの例題参照)。

オプションのstartSelとendSel引数を使用して、object中のテキストを選択できます。startSelとendSelの値はプレーンテキストの選択に使用され、テキスト中のスタイルタグは無視されます。このコマンドの動作はオプションのstartSelとendSel引数に基づき変わります:

- startSelとendSelを省略すると、**ST SET TEXT**はobjectのすべてのテキストをnewTextで置き換えます。
- startSelのみを渡した場合、またはstartSelとendSelが同じ場合、**ST SET TEXT**はnewTextテキストをobjectのstartSelの位置に挿入します。
- startSelとendSel両方渡した場合、**ST SET TEXT**はこれらの引数で指定されたプレーンテキストをnewTextテキストで置き換えます。
- endSelに0をセットすることで、自動的にテキストの最後を指定(startSelに1をセットすることでテキストの最初を指定)できます。

4D では、選択範囲を自動的に指定するために startSel 引数と endSel 引数に渡す定数を提供しています。これらの定数は "Multistyle Text" テーマ内にあります。

定数	型	値	コメント
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。

(*) この定数を使用する際には、object 引数にオブジェクト名を渡す必要があります。フィールドへの参照または変数を渡した場合、コマンドはオブジェクト内の全てのテキストに適用されます。

注: もし startSel 引数が endSel 引数より大きい場合、コマンドは何も行わず、OK変数は0に設定されます。(ただし endSel が0である場合を除く)

システム変数およびセット

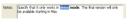
このコマンド実行後、エラーがなければOKシステム変数に1が設定されます。そうでなければ0が設定されます。これは特にスタイルタグが正しく評価できなかった場合に発生します (正しくない、あるいは失われたタグ)。

エラーの場合、変数は変更されません。テキストが評価される際に変数にエラーが発生すると、4Dはテキストをプレーンテキストに変換します。結果、"<"、">"、"&"文字はHTML実体参照に変換されます。

例題 1

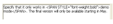
リッチテキストエリア中でユーザーが選択したテキストを変数の内容で置き換えます。

選択されたテキストは以下の通りです:



Specify that it only works in demo **mode**. The final version will only be available starting in May.

フィールドには以下のテキストが格納されています:

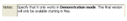


Specify that it only works in <SPAN STYLE="font-weight:bold">demo mode</SPAN>. The final version will only be available starting in May.

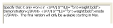
以下のコードを実行すると:

```
vtempo:="Demonstration"
GET HIGHLIGHT([Products]Notes;vStart;vEnd)
ST SET TEXT([Products]Notes;vtemp;vStart;vEnd)
```

フィールドの表示およびその内容は以下のようになります:



Specify that it only works in **Demonstration mode**. The final version will only be available starting in May.



Specify that it only works in <SPAN STYLE="font-weight:bold">Demonstration</SPAN> <SPAN STYLE="font-weight:bold">mode</SPAN>. The final version will only be available starting in May.

例題 2

ST SET PLAIN TEXTコマンドの例題を参照してください

SVG

SVGコマンドの概要
SVG EXPORT TO PICTURE
SVG Find element ID by coordinates
SVG Find element IDs by rect
SVG GET ATTRIBUTE
SVG SET ATTRIBUTE
SVG SHOW ELEMENT

SVGコマンドの概要

SVG (Scalable Vector Graphics) はXMLベースのファイルフォーマット (拡張子は.svg) で、ベクターグラフィックを定義するために使用します。SVGは統計や地図などのデータを公開するために最も一般的に使用されます。

これらのファイルはWebブラウザを使用してネイティブにあるいはプラグイン経由で見ることができます。4DはSVG描画エンジンを含んでいて、SVGファイルをピクチャフィールドや変数に表示できます。SVG EXPORT TO PICTURE コマンドを使用して、SVGの定義に基づくピクチャを4Dで生成できます。またGRAPH コマンドも4Dに統合されたSVGエンジンを使用しています。

このフォーマットに関する詳細は以下のWebサイトを参照してください: http://www.w3.org/Graphics/SVG/.

SVG EXPORT TO PICTURE

SVG EXPORT TO PICTURE (elementRef ; pictVar {; exportType})

引数	型	説明
elementRef	文字	→ ルートXML要素参照
pictVar	ピクチャー	→ XMLツリーを受け取るピクチャ変 (SVG ピクチャ)
exportType	倍長整数	→ 0 = データソースを保持しない 1 = データソースをコピー 2 (デフォルト) = データソースを移動

説明

SVG EXPORT TO PICTURE コマンドはXMLツリーのSVGフォーマットのピクチャを、pictVar 引数で指定したピクチャフィードや変数に保存するために使用できます。

Note: SVGフォーマットに関する詳細はの節を参照してください。

elementRefにはSVGピクチャを含むXMLのルート要素参照を渡します。

pictVarにはSVGピクチャを受け取る4Dのピクチャフィールドまたは変数の参照を渡します。ピクチャはネイティブフォーマットで書き出され、表示される際にはSVG描画エンジンにより再描画されます。

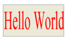
オプションのexportType 引数を使用して、コマンドがXMLデータソースを扱う方法を指定できます。テーマの以下のいずれかの定数を渡すことができます:

定数	型	値	コメント
Copy XML Data Source	倍長整数	1	4Dはピクチャとともに DOMツリーのコピーを保持します。ピクチャをデータベースのピクチャフィールドに保存して、いつでも再び表示したり、書きだしたりできます。
Get XML Data Source	倍長整数	0	4DはXMLデータソースの読み込みのみを行います。XMLデータソースはピクチャと一緒には保持されません。これはコマンドの実行速度を大幅に向上させますが、DOMツリーが保持されないため、ピクチャを格納したり書きだしたりすることはできません。
Own XML Data Source	倍長整数	2	4DはDOMツリーをピクチャとともに書き出します。ピクチャを格納したり書きだしたりでき、かつコマンドの実行も速いです。しかしelementRef XML参照を他の4Dコマンドで使用することはできなくなります。これはexportType 引数が省略された場合のデフォルトモードです。

例題

以下の例題は4Dピクチャに"Hello World"を表示します:

```
C_PICTURE(vpict)
$svg:=DOM Create XML Ref("svg";"http://www.w3.org/2000/svg")
$ref:=DOM Create XML element($svg;"text";"font-size";26;"fill";"red")
DOM SET XML ATTRIBUTE($ref;"y";"1em")
DOM SET XML ELEMENT VALUE($ref;"Hello World")
SVG EXPORT TO PICTURE($svg;vpict;Copy XML Data Source)
DOM CLOSE XML($svg)
```



SVG Find element ID by coordinates

SVG Find element ID by coordinates ({* ;} pictureObject ; x ; y) -> 戻り値

引数	型		説明
*	演算子	→	指定時、pictureObjectはオブジェクト名 (文字列) 省略時、pictureObjectはフィールドまたは変数
pictureObject	ピクチャー	→	オブジェクト名 (* 指定時) または フィーウドまたは変数 (* 省略時)
x	倍長整数	→	X座標 (ピクセル)
y	倍長整数	→	Y座標 (ピクセル)
戻り値	文字	↩	X, Yの位置に見つかった要素のID

説明

SVG Find element ID by coordinatesコマンドは、pictureObject引数で指定されたSVGピクチャ内で、x, y座標により設定された場所で見つかるXML要素のID ("id" または "xml:id" 属性) を返します。このコマンドは特に、SVGオブジェクトを使用してインタラクティブなインタフェースを作成する際に使用します。

Note: SVGフォーマットに関する詳細はXMLユーティリティコマンドの概要を参照してください。

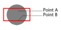
オプションの * 引数を渡すと、pictureObject 引数はオブジェクト名 (文字列) であることを示します。この引数を渡さないと、pictureObject 引数はフィールドまたは変数であることを示します。この場合文字列ではなくフィールドまたは変数の参照 (フィールドまたは変数のみ) を渡します。

ピクチャがフォームに表示されている必要はないことに留意してください。この場合、"オブジェクト名" を使用するシンタックスは使用できず、フィールドまたは変数を渡さなければなりません。

x と y 引数に渡す座標は、ピクチャの左上隅からの相対位置で示されたピクセル数でなければなりません。フォームに表示されたピクチャのコンテキストでは、MouseX と MouseY システム変数の値を使用できます。これらの変数はフォームイベントのOn Clicked と On Double Clicked および On Mouse Enter と On Mouse Move で更新されます。

Note: ピクチャの座標システムにおいて、ピクチャ表示フォーマットにかかわらず、ピクチャがスクロールやズームされていても、MouseXとMouseYは常にピクチャの同じ場所を指し示します (繰り返しフォーマットを除く) 。

使用されるポイントは最初に見つかったポイントです。例えば以下のケースでは、Point Aが渡されると円のIDが返され、Point Bが渡されると四角のIDが渡されます:



座標が重ね合わせまたは複合オブジェクトに対応する場合、コマンドは必要に応じて親要素の間をさかのぼり、有効なID属性を持つ最初のオブジェクトのIDを返します。

コマンドは以下の場合空の文字列を返します:

- ID属性を見つけられないままrootに達した場合。
- 座標がオブジェクトを指していない場合。
- ID属性値が空の文字列の場合。

Note: このコマンドはopacity値 ("fill-opacity" 属性) が0.01より小さいのオブジェクトを検知することはできません。

システム変数およびセット

pictureObjectが有効なSVGピクチャを含んでいない場合、コマンドは空の文字列を返しOKシステム変数に0が設定されます。コマンドが正しく実行されればOKシステム変数に1が設定されます。

SVG Find element IDs by rect

SVG Find element IDs by rect ({* ;} pictureObject ; x ; y ; width ; height ; arrIDs) -> 戻り値

引数	型		説明
*	演算子	→	指定時: pictureObjectはオブジェクト名 (文字) 省略時: pictureObjectは変数
pictureObject	ピクチャー	→	オブジェクト名 (* 指定時) またはフィールドや変数 (* 省略時)
x	倍長整数	→	選択領域の左上の横座標
y	倍長整数	→	選択領域の左上の縦座標
width	倍長整数	→	選択領域の幅
height	倍長整数	→	選択領域の高さ
arrIDs	テキスト配列	←	バインドされた四角が選択領域に交差する要素のID
戻り値	ブール	↩	True = 最低1つの要素が見つかった

説明

SVG Find element IDs by rect コマンドは、バインドされた四角が選択領域に交差するXML要素のID ("id"または"xml:id"属性) をテキストまたは文字配列のarrIDs配列に返します。選択領域はxおよびy引数で指定されます。

最低1つの要素が見つかると、言い換えればarrIDs配列が空でなければ、コマンドはTrueを返します。そうでなければFalseを返します。

このコマンドは特にインタラクティブなグラフィックインタフェースで使用されます。

オプションの * 引数を渡すと、pictureObject引数はオブジェクト名 (文字) です。この引数を渡さないと、pictureObject引数はフィールドまたは変数です。この場合文字列ではなくフィールドや変数の参照 (フィールドまたは変数オブジェクトのみ) を渡します。

ピクチャフィールドや変数で作業をしている場合、コマンドはデータソースに対応するオリジナルのピクチャを使用します。しかしフォームオブジェクトで作業を行っている場合、コマンドはカレントのピクチャを使用します。このピクチャはSVG SET ATTRIBUTEコマンドを使用して変更されているかもしれませんし、フォームオブジェクトのプロパティが保持されています。

xとy引数に渡される座標はピクチャの左上座標 (0, 0) からピクセル単位で表現されます。MouseXとMouseYから返される値を使用できます。これらの変数はOn Clicked、On Double ClickedやOn Mouse EnterとOn Mouse Moveフォームイベントで更新されます。

Note: ピクチャの座標システム中[x;y]は、"繰り返し"フォーマットを除き、ピクチャ表示フォーマットにかかわらず常に同じ場所をポイントします。

バインドされた四角が選択領域に重なるすべての要素が、たとえ他の要素の下になっていても、対象となります。

SVG GET ATTRIBUTE

SVG GET ATTRIBUTE ({* ;} pictureObject ; element_ID ; attribName ; attribValue)

引数	型		説明
*	演算子	→	指定時: pictureObjectはオブジェクト名 (文字) 省略時: pictureObjectは変数
pictureObject	ピクチャー	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
element_ID	テキスト	→	属性値を取得する要素のID
attribName	文字	→	取得する属性
attribValue	文字, 倍長整数	←	現在の属性値

説明

SVG GET ATTRIBUTEコマンドを使用して、オブジェクトまたはSVGピクチャのattribName属性の現在値を取得できます。

オプションの * 引数を渡すと、pictureObject引数はオブジェクト名 (文字) です。この場合、コマンドはオブジェクトにアタッチされた描画イメージの属性値を返します。この値は例えばSVG SET ATTRIBUTEで変更されているかもしれません。
* 引数を渡さないと、pictureObject引数は変数です。従って文字ではなく変数参照 (変数オブジェクトのみ) を渡します。この場合コマンドは、最初に描画されたイメージの属性値を返します (変数のデーターソースに対応)。

注: この原則は既存のSVG Find element ID by coordinatesコマンドにも適用されます。

element_ID 引数は属性値を取得したい要素のID ("id"または"xml:id"属性) を設定するために使用します。

SVG属性に関する詳細は、SVG SET ATTRIBUTEコマンドの説明を参照してください。以下は予約済みまたはアニメーションに関連する4Dの属性です:

属性	**アクセス**	**コメント**
4D-text	読み/書き	テキストノードの内容を置き換え/読み込みます。'text'、'tspan'、および'textArea'要素で利用できます。
4D-bringToFront	読み込み	'true'の場合、ノードを兄弟ノードの前面に移動します。SVG SET ATTRIBUTEコマンドでのみ使用できます。
4D-isOfClass-{IDENT [[S|COMMA] IDENT]*}	読み込み	ノードの継承クラス属性がすべてのクラス名を含む場合に'true'を返します。そうでなければ'false'を返します。例えば"4D-isOfClass-land"に対してノードの継承されたクラスが"land department01" の場合、trueを返します。
4D-enableD2D	読み/書き	'false'の場合、SVG描画エンジンのDirect2Dを無効にします。実際SVGフィルターはDirect2Dでは描画されず、GDI/GDIPlusを使用します。このオプションを使用すればデータベースがDirect2DモードであってもSVGフィルターを利用できるようになります。このオプションはピクチャーがpictureObjectにロード済みである場合のみ効果がある点に留意してください。このオプションはエンジンにグローバルに設定されるため、セッション毎に一回設定すれば十分です (例えばデータベース開始時にテキスト変数からメモリにロードされた小さなSVGのようなケース)。

SVG SET ATTRIBUTE

SVG SET ATTRIBUTE ({* ;} pictureObject ; element_ID ; attrName ; attribValue {; attrName2 ; attribValue2 ; ... ; attrNameN ; attribValueN} {; *})

引数	型	説明
*	演算子	指定時: pictureObjectはオブジェクト名 (文字) 省略時: pictureObjectは変数
pictureObject	ピクチャー	オブジェクト名 (* 指定時) または 変数 またはフィールド(* 省略時)
element_ID	テキスト	1つ以上の属性を設定する要素のID
attrName	文字	指定する属性
attribValue	文字, 倍長整数	属性の新しい値
*	演算子	指定時 = SVG画像の内部DOMツリーを更新 (変数のみ)

説明

SVG SET ATTRIBUTEコマンドは 表示されている画像のSVG描画ツリーまたは画像の内部DOMツリー中で、既存の属性の値を更新するために使用します。

オプションの * 引数を渡すと、pictureObject引数はオブジェクト名 (文字) です。この場合、コマンドはオブジェクトにアタッチされたイメージのパラメーターに適用されます (パラメーターおよびオブジェクトの表示されたイメージは、少なくとも一回**SVG SET ATTRIBUTE**が呼び出されたときのみ作成されることに留意してください)。* 引数を渡さないと、pictureObject引数は変数またはフィールドです。従って文字ではなく変数参照 (変数オブジェクトのみ) またはフィールド参照を渡します。この場合コマンドは、その変数を使用するすべてのオブジェクトに表示されたイメージに適用されます。

デフォルトでは、このコマンドにより行われる更新は描画された画像にのみ適用されます。更新はデータソース (内部DOMツリー) に格納されず、ピクチャーがプログラムで消去されたりフォームが閉じられたりすると、その更新は失われます。しかしpictureObject引数が変数を参照している場合、この更新を画像の内部DOMツリーに転送できます。これを行うには最後の引数として * を渡します。これにより更新をその場で行うことができます。

注:

- 内部DOMツリーへの更新の転送は、pictureObjectがオブジェクトを参照している場合には行えません。
- 更新の転送を可能にするためには、SVG変数が (DOM EXPORT TO VARを使用して) DOMドキュメントから作成されていなければなりません。SVG変数がファイルから作成されている場合、2番目の * 引数を渡すと、データソースには更新可能なDOMドキュメントが含まれないため、コマンドはなにも行わずエラーを生成します。
- SVG画像のデータソースを変更するには4Dが提供するXML DOMやSVGコンポーネントコマンドも使用することができます。

element_ID 引数は、更新したい属性を持つ要素のID ("id"または"xml:id"属性) を指定するために使用します。

attribNameとattribValue引数にはそれぞれ、書き込む属性と値を (変数、フィールド、またはリテラル値で) 渡します。必要なだけ属性/値に組を渡せます。

SVG SET ATTRIBUTEコマンドは、'fill'、'opacity'、'font-family'などほとんどのSVG属性を (追加や削除でなく) 変更するために使用します。SVG属性の完全な説明は、インターネット上のドキュメント (例: http://www.w3.org/TR/SVG11/attindex.html) などを参照してください。表示されるイメージは即座に更新されます。継承されるスタイルの場合、更新は子要素にも適用されます。

技術的な理由で、特定の要素や特定の属性は更新できません。以下の表は更新可能および不可能な要素、さらには変更不可能な属性のリストです:

属性を更新可能な要素

svg	制限: - "width" と "height" は変更できません。(1) - "viewBox" は、"width" と "height"がオリジナルのドキュメントで指定されているときのみ変更できます。
g	
defs	
use	
filter	制限: 子要素 fe_xxx へ変更できません。
circle	
ellipse	
line	
polyline	
polygon	
path	
rect	
text, tspan, textArea	"4d-text" 属性を使用して"text"、"tspan"、および"textArea"要素のテキストを更新します (例題参照)。

属性を変更できない要素

linearGradient, radialGradient, Stop, solidColor, marker, symbol, clipPath, feで始まるフィルターと要素, style, pattern	このグループは参照可能または参照可能な要素に含有可能なすべての要素を示します。つまりこれは、例えばグラデーションの属性を再定義することはできないということを意味します (しかし使用するグラデーションを変更することはできます)。同様に黒のマーカーを赤のマーカーに変更するには、SVGドキュメント内で両マーカー (1つは黒でもう1つは赤) を定義し、どちらかを選択する必要があることを示します。また例えば親要素がシンボルまたはマーカー要素であるとき、四角の色を変えることもできません。

変更できない属性

id or xml:id	
lang or xml:lang	
class or xml:class	
width, height	'svg'要素のの属性のみ(1)

(1) これらの属性は、それらが結果のイメージを定義および構築するため変更できません。*svg*要素の*width*および*height*属性は 4D中で初期のサイズを決定し、ピクチャ作成後このサイズは一定でなければなりません (しかしながら4DのTRANSFORM PICTUREコマンドを使用して結果のピクチャのサイズを変更できます。

SVG GET ATTRIBUTEコマンドの説明ではアニメーション用に予約された4D属性のリストを参照できます。

サポートされていない要素の属性やその子要素を更新しようとすると、コマンドはなにも行わず、エラーが生成されます。

コマンドがフォームのコンテキストで実行されないか無効なpictureObjectが渡された場合、*OK* 変数に0が設定されます。コマンドが正しく実行されると1が設定されます。

例題

テキスト型の要素の内容を更新する:

```
SVG SET ATTRIBUTE(*;picture_object_name;text_element_ID;"4d-text";"This is a text")
```

Note: 衝突の恐れなしにCSSスタイルシート内で属性を使用するため、名前空間がありません。

SVG SHOW ELEMENT

SVG SHOW ELEMENT ({* ;} pictureObject ; id {; margin})

引数	型		説明
*	演算子	⇒	指定時: pictureObjectはオブジェクト名 (文字) 省略時: pictureObjectは変数
pictureObject	ピクチャー	⇒	オブジェクト名 (* 指定時) または 変数またはフィールド (* 省略時)
id	テキスト	⇒	表示する要素のID属性
margin	倍長整数	⇒	表示のマージン (デフォルトでピクセル単位)

説明

SVG SHOW ELEMENT コマンドは、id引数で指定した"id"属性を持つ要素を表示するように、pictureObject SVGドキュメントを移動します。.

オプションの * 引数を渡すと、pictureObject引数はオブジェクト名 (文字) です。この場合、コマンドはオブジェクトにアタッチされた描画ピクチャに適用されます。この引数を渡さないと、pictureObject引数は変数やフィールドであり、変数参照 (変数オブジェクトのみ) またはフィールド参照を渡します。この場合この変数を使用するすべてのオブジェクトに描画されたピクチャに適用されます (最初に描画されたピクチャを除きます)。

コマンドはSVGドキュメントを移動し、境界がバインドされた四角で定義されるすべてのオブジェクトが表示されるようにします。margin引数を使用して、ドキュメントの縁からの、オブジェクトを表示するマージンを指定できます。言い換えればバインドした四角はmarginピクセルだけ高さと幅が大きくなります。デフォルトで移動値は4ピクセルです。

このコマンドはスクロールバー付きの"左上"表示モードのみで効果があります。

フォームのコンテキストでコマンドが実行されていないか、無効な pictureObject が渡された場合、*OK* 変数に0が設定されます。コマンドが正しく実行されると1が設定されます。

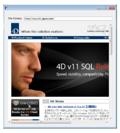
Webエリア

Webエリアのプログラムによる管理
WA Back URL available
WA Create URL history menu
WA Evaluate JavaScript Updated 14.0
WA EXECUTE JAVASCRIPT FUNCTION Updated 14.0
WA Forward URL available
WA Get current URL
WA GET EXTERNAL LINKS FILTERS
WA Get last filtered URL
WA GET LAST URL ERROR
WA Get page content
WA Get page title
WA GET PREFERENCE
WA GET URL FILTERS
WA GET URL HISTORY
WA OPEN BACK URL
WA OPEN FORWARD URL
WA OPEN URL
WA REFRESH CURRENT URL
WA SET EXTERNAL LINKS FILTERS
WA SET PAGE CONTENT
WA SET PAGE TEXT LARGER
WA SET PAGE TEXT SMALLER
WA SET PREFERENCE Updated 14.0
WA SET URL FILTERS
WA STOP LOADING URL

Webエリアのプログラムによる管理

このテーマのコマンドは、Webエリアタイプのフォームオブジェクトをプログラミングによって管理するために使用します。

Webエリアはあらゆるタイプのwebコンテンツを4D環境で表示します。静止画や動画を含むHTMLページ、ファイル、ピクチャ、Javascript、Flash、PDFそしてMS Officeドキュメント (MS OfficeがインストールされているWindowsのみ) などです。以下のピクチャはフォームに格納されているWebエリアで、HTMLページを表示しています。


Site Address: http://www.4d-japon.com/
When the solution matters
Product Lines
Solutions
Corporate
4D v11 SQL Rele
Speed, stability, compatibility. Fo
Users (501)
4D Server v11 SQL Breaks the 500 Client Barrier
4D News
4D v11 SQL Release 3 (11.3) 公開開始!

Webエリアテーマのコマンドに加え、いくつかの標準アクションやフォームイベントを使用して、デベロッパはこれらのWebエリアの機能をコントロールすることができます。特定の変数を用いると、エリアと4D環境の間での情報交換が可能となります。つまり、これらのツールを利用すると、フォーム上にベーシックなWebブラウザを構築することができるのです。

Webエリアを作成して接続する

Webエリアは、4Dフォームエディタのオブジェクトバーにあるプラグインエリア/サブフォームボタンに追加された新しい項目を使用して作成されています (詳細はデザインリファレンスマニュアルのWebエリアを参照してください) 。

他の動的なフォームオブジェクトのように、Webエリアはオブジェクト名とテキスト型の変数名を持ちます。これらはWebエリアをプログラミングによって処理する際に使用されます。 特にWebエリアに対してOBJECT SET VISIBLEコマンドとOBJECT MOVEコマンドを使用することができます。

注: Webエリアに割り当てられるテキスト変数は参照を格納していません。そのためメソッドの引数として渡すことはできません。例えばWebエリアにMyAreaという変数名が割り当てられているとき、以下のコードを使用することはできません。

```
Mymethod(MyArea)
```

Mymethodコード

```
WA REFRESH CURRENT URL($1) // 動作しない
```

このタイプのプログラミングに対しては、ポインタを使用する必要があります。

```
Mymethod(->MyArea)
```

Mymethodコード

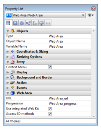
```
WA REFRESH CURRENT URL($1->) // 動作する
```

割り当てられる変数を管理する

標準的なオブジェクト変数に加え (前述の項目を参照) 、指定された2つの変数が、自動的にそれぞれのWebエリアに割り当てられます。

- URL変数
- 進捗変数

これらの変数はそれぞれ、デフォルトでWebエリア_url と Webエリア_progressという名前が付けられます。必要であれば、これらの名前を変更することも可能です。これらの変数はプロパティリストからアクセスできます。



Type	Web Area
Object Name	Web Area
Variable Name	Web Area
Context Menu	☑
URL	Web Area_url
Progression	Web Area_progress
Use integrated Web Kit	☑
Access 4D methods	☑

URL変数

URL変数は文字列タイプです。この変数にはWeb エリアにロードされたURL またはロード中のURL が格納されます。

変数とWeb エリア間の連携は双方向で行われます。

- ユーザが新しいURL を変数に割り当てると、このURLは自動でWeb エリアにロードされます。
- Web エリアでブラウズが行われると、自動で変数の内容が更新されます。このエリアはブラウザのアドレスバーのように機能します。Web エリアの上側にテキストエリアを置いて、内容を表示させることができます。

URL 変数とWA OPEN URL コマンド

URL 変数はWA OPEN URLコマンドと同じ効果をもたらします。しかしながら以下の違いに注意してください。

- ドキュメントにアクセスする場合、この変数はRFC 準拠 ("file://c:/My%20Doc") なURL のみを受け付け、システムパス名 ("c:\MyDoc") は受け付けません。WA OPEN URL コマンドは両方の記法を受け付けます。
- URL 変数が空の文字列の場合、Web エリアはURL をロードしません。WA OPEN URL コマンドはこの場合エラーを生成します。
- URL 変数にプロトコル (http, mailto, file など) を含まない場合、Web エリアは"http://" を付加します。WA OPEN URL コマンドは付加しません。
- Web エリアがフォーム上で表示されていない場合 (フォームの他のページにWeb エリアがある場合等)、WA OPEN URL コマンドを実行しても効果はありません。一方、URL 変数に値を代入すると、カレントのURL が更新されます。

進捗変数

進捗変数は倍長整数タイプです。この変数には0 から100 までの値が格納され、この数値はWeb エリアに表示されるページのロードされたパーセンテージを表します。
この変数は4D が自動で更新します。手動で変更することはできません。

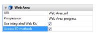
4Dメソッドへのアクセス

Webエリアで実行されるJavaScripeコードから4Dメソッドを呼び出して、戻り値を取得することができます。

重要: この機能はWebエリアがレンダリングエンジンとして統合されたWeb Kitを使用している場合に限り、使用可能です。

Webエリアの設定

4DメソッドをWebエリアから呼び出せるようにするためには、エリアのプロパティリスト内の、**4Dメソッドコールを許可** のオプションにチェックをする必要があります:



URL	Web Area_url
Progression	Web Area_progress
Use integrated Web Kit	☑
Access 4D methods	☑

注: このオプションは、**統合されたWeb Kitを使用** のオプションにチェックをしている場合のみ有効です。

このプロパティがチェックされている場合、特別なJavaScriptオブジェクト($4d)がWebエリア内に表示され、これを使用して4Dプロジェクトメソッドの呼び出しを管理することが出来るようになります。

$4Dオブジェクトの使用

4Dメソッドコールを許可 のオプションにチェックがされている場合、4Dの統合Web Kitは、"."オブジェクト記法を使用することによって4Dプロジェクトメソッドと使用できる$4dというJavaScriptオブジェクトをエリアに提供します。

例えば、HelloWorldという4Dメソッドを呼び出す場合には、以下の宣言を実行するだけです:

```
$4d.HelloWorld();
```

警告: JavaScriptは大文字小文字を区別するので、この場合、オブジェクトの名前は$4d (dは小文字)であることに注意して下さい。

4Dメソッドへの呼び出しのシンタックスは以下の様になります:

```
$4d.4DMethodName(param1,paramN,function(result,error){})
```

- param1...paramN: 4Dメソッドに対して必要なだけ引数を渡すことが出来ます。
  これらの引数は、JavaScriptにサポートされている型であればどんな方でも渡す事が出来ます(文字列、数値、配列、オブジェクト)。

- function(result,error): 最後の引数として渡される関数。この"コールバック"関数は、4Dメソッドが実行を終えると同時に呼び出されます。この関数はふたつの引数を受け取ります:
  - result: 4Dメソッドの実行の返り値です。"$0"という4D式の中に返されます。
    返り値はJavaScriptでサポートされている型(文字列、数値、配列、オブジェクト)のどれかになります。新しい C_OBJECT コマンドを使用して、オブジェクトを返すこともできます。
    Note: デフォルトとして、4DはUTF-8文字コードで動作しています。拡張された文字(アクセントが付いた文字など)を含むテキストを返す場合には、Webエリアで表示されるページの文字コードがUTF-8に宣言されていることを確認して下さい。文字コードがUTF-8でない場合、文字が正しく表示されない可能性があります。この場合、以下の1行をHTMLページに追加して文字コードを宣言して下さい:
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  - error (任意): エラースタックが格納された、オブジェクトの配列です。この配列は、4D側で(ON ERR CALL を使用して)エラーを管理しなかった場合にのみ作成されます。
    それぞれのオブジェクトは三つのプロパティから構成されます
    - *message* (テキスト): エラーの詳細
    - *errCode* (数字): エラーコード
    - *componentSignature* (テキスト): エラーの発生場所を示す内部コンポーネントの識別コード(4文字) 例:"dbmg"はデータベースエンジンをさします。

例題 1

today という名の4Dプロジェクトメソッドがあり、そのメソッドは引数を受け付けず、カレントの日付を文字列として返す場合について考えてみます。

today メソッドの4D コードは以下の様になります:

```
C_TEXT($0)
$0:=String(Current date;System date long)
```

Webエリアでは、4D メソッドは以下のシンタックスで呼び出し可能です:

```
$4d.today()
```

この4Dメソッドは引数を何も受け取りませんが、$0の値を、4Dによって呼び出されたコールバック関数へとメソッドの実行後に返します。
WebエリアによってロードされたHTMLページ内に日付を表示したい場合を考えます。

HTMLページのコードは以下の様になります:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

</head>
<body>Today is: <div id="madiv"></div>
</body>
</html>
```

例題 2

calcSum という4Dプロジェクトメソッドがあり、そのメソッドが($1...$n)という引数を受け取り、その合計を$0に返すという場合について考えます:

calcSum メソッドの4D コードは以下の様になります:

```
C_REAL(${1}) // n個の実数型の引数を受け取ります。
C_REAL($0) // 実数の値を返します。
C_LONGINT($i;$n)
$n:=Count parameters
For($i;1;$n)
   $0:=$0+${$i}
End for
```

Web エリア内で実行されるJavaScriptコードは以下の様になります:

```
$4d.calcSum(33, 45, 75, 102.5, 7, function(dollarZero)
    {
        var result = dollarZero // result is 262.5
    });
```

例題 3

ここでは、例2で登場した calcSum メソッドに、エラーハンドラーを追加したものを使用する場合を考えます:

```
$4d.calcSum("alpha", 45, 75, 102.5, 7, function(result, error)
    {
        if (error != null)
            alert( "got error: " + error[0].message);
        else
            alert( "got result: " + result);
    });
```

フォームイベント

特定のフォームイベントは、Webエリアをプログラミングで管理するこを目的としています。すなわち、リンクの起動に関連しています:

- On Begin URL Loading
- On URL Resource Loading
- On End URL Loading
- On URL Loading Error
- On URL Filtering
- On Open External Link
- On Window Opening Denied

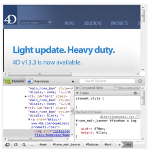
更に、Webエリアは以下の汎用フォームイベントをサポートしています:

- On Load
- On Unload
- On Getting Focus
- On Losing Focus

これらのイベントに関する詳細は、Form event コマンドの記述を参照してください。

Webインスペクターへのアクセス

4D v14では、フォームの Web エリア内で Web インスペクターを見たり使用したりすることが出来るようになりました。Web インスペクターは統合された Web Kitデバッガーであり、Webページの情報の、コードとフローを解析します。この Web インスペクターはChromeのブラウザに含まれているものと基本的に同じものです



Web インスペクターの表示

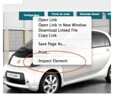
Web エリア内でWebインスペクターを表示させるためには、以下の手順に従ってください:

- Webエリアの、統合されたWeb Kitのオプションを選択して下さい(Web インスペクターを使用するためにはこのオプションを選択する必要があります。統合されたWebKitを使用する を参照して下さい)。
- エリア内のコンテキストメニューを有効化して下さい(インスペクターを使用するためにはメニューを使用する必要があります。コンテキストメニュー を参照して下さい)。
- 以下の宣言をWebエリア内で行い、インスペクターの使用を明示的に有効化して下さい:
  `WA SET PREFERENCE(*;"WA";WA enable Web inspector;True)`

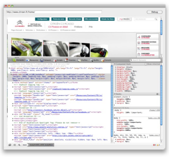
  詳細な情報に関しては、WA SET PREFERENCE コマンドを参照して下さい。

Web インスペクターの使用

上記の手順を踏んで設定を完了すると、エリア内でコンテキストメニューを開くと**要素の詳細を表示**という新しいオプションが追加されているはずです:



このオプションを選択すると、Webエリアは二つに分割され、Webインスペクターは下側のパネル内に表示されます:



また、インスペクターは別のウィンドとして表示して使用することもできます。エリアの左下にある。"Undock in separate windows" ボタンをクリックして下さい:

Web インスペクターは、Web Kit レンダリングエンジンに含まれています。このデバッガーの機能の詳細に関しては、以下のアドレスにてGoogleより提供されているドキュメントを参照して下さい: https://developers.google.com/chrome-developer-tools/docs/scripts

Webエリアの利用に際する注意点

ユーザインタフェース

フォームが実行されると、他のフォームエリアと対話することを可能にする、標準のブラウザインタフェース機能がWeb エリア中で利用可能になります。

- **編集** メニューコマンド: Web エリアにフォーカスがあるとき、選択された内容に基づき、**編集** メニューコマンドを使用してコピーやペースト、すべてを選択などのアクションを実行できます。
- コンテキストメニュー: Web エリアで、システムの標準コンテキストメニューを使用できます (コンテキストメニューの表示はWA SET PREFERENCE コマンドを使用してコントロールできます) 。
- ドラッグ＆ドロップ: 4D のオブジェクトプロパティに基づき、ユーザはWeb エリア上で、またはWeb エリアと4D フォームオブジェクト間で、テキストやピクチャ、ドキュメントをドラッグ＆ドロップできます。

MS Officeドキュメント (Windows)

Windows では、Web エリアを使用して、Microsoft Office ドキュメントを表示したり、編集したりできます (Microsoft Office がマシンにインストールされている場合)。特にWord、Excel、PowerPoint ドキュメントなど (.doc、.xls、.ppt などの拡張子) を処理できます。MS Office XMLフォーマットもサポートされています。

Note: MS Office 2007 はデフォルトでWeb ブラウザでのドキュメントの表示を許可していません。ドキュメントは常に新規ウィンドウに表示されます。以下のアドレスで説明されている方法で、この動作を変更できます。
http://support.microsoft.com/kb/162059/ja-jp
最後に、Windowsでは、Web エリアを使用して、ローカルや外部のフォルダを表示できます (ftp:// プロトコルやネットワークパス名 \\myserver\myvolume を使用して)。

WebエリアとWebサーバのコンフリクト (Windows)

Windows では、Web エリアから、同じ4D アプリケーションで起動されているWeb サーバへのアクセスはお勧めできません。これを行うとコンフリクトが発

生し、アプリケーションがフリーズすることがあります。もちろん他の4D から4D Server のWeb エリアにアクセスすることはできます。自身のWeb サーバにアクセスできないということです。

プロトコルの挿入 (Mac OS)

Mac OS 上のWeb エリアで、プログラムにより処理されるURL はプロトコルで開始されていなければなりません。つまり"www.mysite.com" ではなく"http://www.mysite.com" 文字列を渡さなければならないということです。

WA Back URL available

WA Back URL available ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
戻り値	ブール	←	開かれた一連のURLで、前のURLが有効ならTrue、そうでなければFalse

説明

WA Back URL available コマンドは、* と object 引数で指定したWebエリアに開かれた一連のURLで、前のURLが利用できるかどうかを知るために使用します。

コマンドはURLがあればTrueを、なければFalseを返します。このコマンドは特に、ナビゲーションボタンを有効/無効にするために使用します。

WA Create URL history menu

WA Create URL history menu ({* ;} object {; direction}) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
direction	倍長整数	→	0 または省略=戻るURLのリスト, 1=進むURLリスト
戻り値	MenuRef	↩	メニュー参照

説明

WA Create URL history menu コマンドは、* と object で指定したWebエリアがセッション中に訪問したURL間をナビゲートするメニューを作成します。このコマンドを使用して、カスタムのナビゲーションインタフェースを作成できます。

提供される情報はセッションに限定されます。言い換えればナビゲーションは同じWebエリアで、フォームが閉じられない間実行されます。

directionには取得するリストを指定する値を渡します。""テーマの以下の定数を使用できます:

定数	型	値
WA next URLs	倍長整数	1
WA previous URLs	倍長整数	0

direction 引数を省略すると、0 が使用されます。

メニューが生成されたら、4DのDynamic pop up menuコマンドで表示し、4Dの標準メニュー管理コマンドを使用して処理できます。Dynamic pop up menu コマンドから返される文字列には、訪問したページのURLが含まれます (例題参照)。

メニューを使用しなくなったら、RELEASE MENU コマンドを呼び出してURL履歴メニューを削除します。

例題

以下のコードを、ポップアップメニューを持つ"戻る"3Dボタンに関連付けることができます:

```
Case of
  `シングルクリック
   :(Form event=On Clicked)
      WA OPEN BACK URL(WA_area)
  `矢印のクリック -> ポップアップ表示
   :(Form event=On Arrow Click)
  `戻る履歴メニューを作成
      $Menu:=WA Create URL history menu(WA_area;WA previous URLs)
  `ポップアップにこのメニューを表示
      $URL:=Dynamic pop up menu($Menu)
  `項目が選択されたら
      If($URL#"")
  `Webページを開く
         WA OPEN URL(WA_area;$URL)
      End if
  `メニューを削除してメモリを解放
      RELEASE MENU($Menu)
End case
```

WA Evaluate JavaScript

WA Evaluate JavaScript ({* ;} object ; jsCode {; type}) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
jsCode	文字	→	JavaScriptコード
type	倍長整数	→	Type into which to convert result
戻り値	Object, テキスト, ポインター, 実数, 日付, 時間	↩	実行結果

説明

WA Evaluate JavaScript コマンドは、* と object 引数で指定したWebエリアで、jsCodeに渡されたJavaScriptコードを実行し、結果を戻します。

デフォルトでは、コマンドは値を文字列として返します。任意の type 引数を用いることによって、戻ってくる値の型を指定することができます。使用できる定数は "Field and Variable Types" テーマ内にある以下の一覧の通りです。

定数	型	値
Is Boolean	倍長整数	6
Is Date	倍長整数	4
Is Integer	倍長整数	8
Is Integer 64 bits	倍長整数	25
Is JSON null	倍長整数	255
Is LongInt	倍長整数	9
Is object	倍長整数	38
Is Real	倍長整数	1
Is String Var	倍長整数	24
Is Text	倍長整数	2
Is Time	倍長整数	11
Object array	倍長整数	39

重要: 複雑な型(オブジェクト)の結果を扱うことが出来るのは、WebエリアがWeb Kitをレンダリングエンジンとして使用している場合に限られます。

例題 1

この例題のJavaScriptコードは、履歴中前のURLを表示します:

```
$result:=WA Execute JavaScript(MyWArea;"history.back()")
```

例題 2

以下に返ってきた結果の変換を含む例を紹介します。

HTML ファイル内にあるJavaScriptファンクションの例:

```
<!DOCTYPE html>
<html>
    <head>
        <script>
        function evalLong(){
            return 123;
        }
        function evalText(){
            return "456";
        }
        function evalObject(){
            return {a:1,b:"hello world"};
        }
        function evalDate(){
            return new Date();
        }
    </script>
    </head>
    <body>
        TEST PAGE
    </body>
</html>
```

4Dフォームメソッドでは以下の様に記述します:

```
 If(Form event=On Load)
    WA OPEN URL(*;"Web Area";"C:\\myDatabase\\index.html")
 End if
```

その後、4DからのJavaScriptコードを以下の様に評価します:

```
 $Eval1:=WA Evaluate JavaScript(*;"Web Area";"evalLong()";Is LongInt))
  //$Eval1 = 123
  //$Eval1 = 型が省略されていた場合は"123"
 $Eval2:=WA Evaluate JavaScript(*;"Web Area";"evalText()";Is String Var)
  //$Eval2 = "456"
 $Eval3:=WA Evaluate JavaScript(*;"Web Area";"evalObject()";Is object)
  //$Eval3 = {"a":1,"b":"hello world"}
 $Eval4:=WA Evaluate JavaScript(*;"Web Area";"evalDate()";Is Date)
  // $Eval4 = 06/21/13
  // $Eval4 = 型が省略されていた場合は"2013-06-21T14:45:09.694Z"
```

WA EXECUTE JAVASCRIPT FUNCTION

WA EXECUTE JAVASCRIPT FUNCTION ({* ;} object ; jsFunction ; result|* {; param}{; param2 ; ... ; paramN})

引数	型	説明
*	演算子	→ 指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→ オブジェクト名 (* 指定時) または 変数 (* 省略時)
jsFunction	文字	→ 実行するJavaScript関数名
result|*	変数	→ または関数が結果を返さない場合 * ← 関数結果 (返される場合)
param	文字, Number, 日付, Object	→ 関数に渡す引数

説明

WA EXECUTE JAVASCRIPT FUNCTION コマンドは、* と object で指定したWebエリアで、jsFunctionに渡したJavaScript関数を実行し、result 引数に結果を返します (オプション)。

関数が結果を返さない場合、* in the result 引数に * を渡します。

paramには関数の引数を含む引数を1つ以上渡せます。

コマンドは、入力 (param) と出力 (result) において複数の型の引数をサポートします。数値、日付、オブジェクト、そして文字列型の渡したり受け取ったりすることが出来ます。

重要: 複雑な型(オブジェクト、日付等)の結果を扱うことが出来るのは、WebエリアがレンダリングエンジンにI統合Web Kitを使用している場合に限られます。

例題 1

3 つの引数を使用してJavaScript関数を呼び出す:

```
$JavaScriptFunction:="TheFunctionToBeExecuted"
$Param1:="10"
$Param2:="true"
$Param3:="1,000.2" `注:千区切りは","で、小数点は "."

WA EXECUTE JAVASCRIPT FUNCTION(MyWArea;$JavaScriptFunction;$Result;$Param1;$Param2;$Param3)
```

例題 2

"getCustomerInfo" という JavaScript ファンクションは、番号ID を引数として受け取り、結果をオブジェクトとして返します:

```
C_OBJECT($Result)
C_LONGINT($ID)
$ID:=1000
WA EXECUTE JAVASCRIPT FUNCTION(*,"WA";"getCustomerInfo";$Result;$ID)
```

WA Forward URL available

WA Forward URL available ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
戻り値	ブール	↩	開かれた一連のURLで、次のURLが有効ならTrue、そうでなければFalse

説明

WA Forward URL available コマンドは、*と object 引数で指定したWebエリアに開かれた一連のURLで、次のURLが利用できるかどうかを知るために使用します。

コマンドはURLがあればTrueを、なければFalseを返します。このコマンドは特に、ナビゲーションボタンを有効/無効にするために使用します。

WA Get current URL

WA Get current URL ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
戻り値	文字	↩	現在Webエリアにロードされているurl

説明

WA Get current URL コマンドは、* と object 引数で指定したWebエリアに現在表示されているページのURLアドレスを返します。

現在のURLが利用できない場合、コマンドは空の文字列を返します。

Webページが完全にロードされると、この関数から返される値はWebエリアに関連付けられたURL変数の値と同じです。ページがロード中の場合、2つの値は異なります。関数は完全にロードされたURLを返し、変数にはロード中のURLが格納されています。

例題

"www.apple.com"のページが表示されていて"www.4d.com"ページをロード中の場合:

```
$url:=WA Get current URL(MyWArea) `は "http://www.apple.com"
  `関連付けられたURL変数は "http://www.4d.com"
```

WA GET EXTERNAL LINKS FILTERS

WA GET EXTERNAL LINKS FILTERS ({* ;} object ; filtersArr ; allowDenyArr)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
filtersArr	文字配列	←	フィルタ配列
allowDenyArr	ブール配列	←	許可-拒否配列

説明

WA GET EXTERNAL LINKS FILTERS コマンドはfiltersArr と allowDenyArr配列に、* と object 引数で指定したWebエリアの外部リンクフィルタを返します。フィルタが有効でない場合、空の配列が返されます。

フィルタはWA SET EXTERNAL LINKS FILTERS コマンドでインストールされます。セッション中に配列が最初期化されても、WA GET EXTERNAL LINKS FILTERS コマンドを使用すれば現在の設定を取得できます。

WA Get last filtered URL

WA Get last filtered URL ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
戻り値	文字	↩	最後にフィルタされたURL

説明

WA Get last filtered URL コマンドは、*と object 引数で指定したWebエリアで、最後にフィルタされたURLを返します。

URLは以下のいずれかの理由でフィルタされることがあります:

- URLがフィルタにより拒否された (WA SET URL FILTERS コマンド)。
- デフォルトブラウザでリンクが開かれる (WA SET EXTERNAL LINKS FILTERS コマンド)。
- URLがポップアップウィンドウを開こうとしている。

フィルタされたURLを知るためには、On URL Filtering、On Open External Link、そしてOn Window Opening Deniedフォームイベントのコンテキストでこのコマンドを呼び出すことをお勧めします。

WA GET LAST URL ERROR

WA GET LAST URL ERROR ({* ;} object ; url ; description ; errorCode)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
url	文字	←	エラー元のURL
description	文字	←	エラーの説明 (Mac OS)
errorCode	倍長整数	←	エラーコード

説明

WA GET LAST URL ERROR コマンドを使用して、* と object 引数で指定したWebエリアで発生した最後のエラーに関する情報のいくつかの項目を取得できます。

この情報は3つの変数に返されます:

- url: エラーを発生させたURL。
- description (Mac OS のみ): エラーについての説明 (利用可能な場合)。エラーにテキストが関連付けられていない場合、空の文字列が返されます。Windowsでは常に空の文字列が返されます。
- errorCode: エラーコード。
  - コードが400以上の場合、それはHTTPプロトコル関連のエラーです。このタイプのエラーに関する詳細情報は、以下のアドレスを参照してください:
  http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
  - 上記以外の場合、WebKit (Mac OS) または ActiveX (Windows) から返されたエラーです。

発生したエラーの原因を知るためには、On URL Loading Errorフォームイベントのフレームワークでこのコマンドを呼び出すことをお勧めします。

WA Get page content

WA Get page content ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
戻り値	文字	↩	HTMソースコード

説明

WA Get page content コマンドは、*と object 引数で指定したWebエリアの現在のページまたは表示されているページのHTMLコードを返します。

現在のページの内容を利用できない場合、このコマンドは空の文字列を返します。

WA Get page title

WA Get page title ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
戻り値	文字	↩	現在のページのタイトル

説明

WA Get page title コマンドは、*と object 引数で指定したWebエリアの現在のページまたは表示されているページのタイトルを返します。タイトルはHTMLのtitleタグに対応します。

現在のURLでタイトルが利用できない場合、このコマンドは空の文字列を返します。

WA GET PREFERENCE

WA GET PREFERENCE ({* ;} object ; selector ; value)

引数	型		説明
*	演算子	→	指定されると、オブジェクトがオブジェクト名 (文字列) 省略されると、オブジェクトは変数
object	フォームオブジェクト	→	オブジェクトの名前 (引数 * が指定されると) または、変数 (引数 * が省略されると)
selector	倍長整数	→	取得する環境設定
value	変数	←	環境設定のカレント値

説明

WA GET PREFERENCE コマンドを使用して、* と object によって指定されたWebエリアの環境設定の現在値を取得します。

取得したい値を持つ環境設定を引数 selector に渡します。この場合、" Web Area " テーマにある以下の定数の1つを渡します。

定数	型	値	コメント
WA enable contextual menu	倍長整数	4	Webエリア内で標準のコンテキストメニューの表示を許可する
WA enable Java applets	倍長整数	1	Webエリア内でJava appletの実行を許可する
WA enable JavaScript	倍長整数	2	Webエリア内でJavaScriptコードの実行を許可する
WA enable plugins	倍長整数	3	Webエリア内でプラグインのインストールを許可する
WA enable Web inspector	倍長整数	100	Web エリア内でインスペクターの表示を許可する

これらの環境設定に関する詳細は、WA SET PREFERENCE コマンドの記述を参照してください。

引数 value には、環境設定の現在値を受け取る変数を渡します。変数のタイプは環境設定によって異なります。value 変数は必ずブール型です。
設定がアクティブであれば **True** が、それ以外の場合には **False** が格納されます。

WA GET URL FILTERS

WA GET URL FILTERS ({* ;} object ; filtersArr ; allowDenyArr)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
filtersArr	文字配列	←	フィルタ配列
allowDenyArr	ブール配列	←	許可-拒否配列

説明

WA GET URL FILTERS コマンドはfiltersArr と allowDenyArr配列に、* と object 引数で指定したWebエリアで有効なフィルタを返します。フィルタが有効でない場合、空の配列が返されます。

フィルタはWA SET URL FILTERS コマンドでインストールされます。セッション中に配列が最初期化されても、WA GET URL FILTERS コマンドを使用すれば現在の設定値を知ることができます。

WA GET URL HISTORY

WA GET URL HISTORY ({* ;} object ; urlsArr {; direction {; titlesArr}})

引数	型	説明
*	演算子	→ 指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→ オブジェクト名 (* 指定時) または 変数 (* 省略時)
urlsArr	文字配列	← 訪問したURLの配列
direction	倍長整数	→ 0または省略=前のURL配列, 1=次のURL配列
titlesArr	文字配列	← ウィンドウタイトルの配列

説明

WA GET URL HISTORY コマンドは、* と object 引数で指定したWebエリアのセッション中に訪問したURLを含む1つまたは2つの配列を返します。このコマンドを使用して、カスタムのナビゲーションシステムを作成できます。

提供される情報はセッションに限定されます。言い換えればナビゲーションは同じWebエリアで、フォームが閉じられない間実行されます。

urlsArr 配列には、訪問したURLが返されます。direction 引数が渡されればその値に基づき、配列に前のURL (デフォルト動作)、または次のURLが返されます。これらのリストは、ブラウザの標準の次ページや前ページの内容に対応します。

URLは時間順に並びかえられます。

directionには取得するリストを指定する値を渡します。""テーマの以下の定数を使用できます:

定数	型	値
WA next URLs	倍長整数	1
WA previous URLs	倍長整数	0

direction 引数を省略すると、0が使用されます。

titlesArr 引数を渡すと、URLに関連付けられたウィンドウの名前が返されます。この配列はurlsArr 配列と同期しています。

WA OPEN BACK URL

WA OPEN BACK URL ({* ;} object)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)

説明

WA OPEN BACK URL コマンドは、* と object 引数で指定したWebエリアに開かれた一連のURL中、現在のURLのひとつ前のURLをロードします。

前のURLがない場合、コマンドは何も行いません。前のURLが有効かどうかは、WA Back URL available コマンドで知ることができます。

WA OPEN FORWARD URL

WA OPEN FORWARD URL ({* ;} object)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)

説明

WA OPEN FORWARD URL コマンドは、* と object 引数で指定したWebエリアに開かれた一連のURL中、現在のURLのひとつ次のURLをロードします。

次のURLがない場合、コマンドは何も行いません。前のURLが有効かどうかは、WA Forward URL available コマンドで知ることができます。

WA OPEN URL

WA OPEN URL ({* ;} object ; url)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
url	文字	→	WebエリアにロードするURL

説明

WA OPEN URLコマンドは、url 引数に渡したURLを、* と object 引数で指定したWebエリアにロードします。

urlに空の文字列を渡すと、**WA OPEN URL**コマンドは何も行わず、エラーも生成されません。Webエリアに空のページをロードするには、url引数に"about:blank"文字列を渡します。

既存のOPEN WEB URLコマンドのように、**WA OPEN URL**はファイルを指定するための複数のシンタックスをurl 引数に受け入れます:

- POSIXシンタックス: "file:///c:/My%20File"
- システムシンタックス: "c:\MyFolder\MyFile" (Windows) または "MyDisk:MyFolder:MyFile" (Mac OS).

注: 互換性のため (2つのスラッシュを使用する) "file://" シンタックスを4Dは受け入れますが、これはRFCに準拠していません。RFCに準拠した"file:///"シンタックス (3つのスラッシュ "/") 利用を推奨します。

このコマンドは、Webエリアに関連付けられた"URL"変数の値を更新することと同じ効果があります。例として、エリアのURL変数がMyWArea_urlのとき:

```
MyWArea_url:="http://www.4d.com/"
```

は以下と同じです:

```
WA OPEN URL(MyWArea;"http://www.4d.com/")
```

WA REFRESH CURRENT URL

WA REFRESH CURRENT URL ({* ;} object)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)

説明

WA REFRESH CURRENT URL コマンドは、* と object 引数で指定したWebエリアに現在表示されているURLを再読み込みします。

WA SET EXTERNAL LINKS FILTERS

WA SET EXTERNAL LINKS FILTERS ({* ;} object ; filtersArr ; allowDenyArr)

引数	型	説明
*	演算子	→ 指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→ オブジェクト名 (* 指定時) または 変数 (* 省略時)
filtersArr	文字配列	→ フィルタ配列
allowDenyArr	ブール配列	→ 許可-拒否配列

説明

WA SET EXTERNAL LINKS FILTERS コマンドを使用して、* と object 引数で指定したWebエリアの外部リンクフィルタを設定できます。外部リンクフィルタは、リンクを使用して現在のページに関連付けられているURLをWebエリアで開くか、マシンのデフォルトWebブラウザで開くかどうか決定するために使用されます。

ユーザが現在のページでリンクをクリックすると、4D は外部リンクフィルタのリストを照会し、リクエストされたURL をマシンのブラウザで開くかチェックします。開く場合、URL に対応するページがWeb ブラウザで開かれ、On Open External Linkフォームイベントが生成されます。そうでなければ (デフォルト動作)、URL に対応するページはWeb エリア内に表示されます。URL の判定はfiltersArrとallowDenyArr 配列の内容に基づき行われます。

filtersArr と allowDenyArr 配列は同期されていなければなりません。

- filtersArr 配列のそれぞれの要素には、フィルタするURL が含まれます。1つ以上の文字を表すワイルドカードとして * を使用できます。
- allowDenyArr配列のそれぞれ対応する要素には、URL をWeb エリアで表示する (True) かWeb ブラウザで表示する (False) かを示すブール値が含まれます。

同じURL が許可および拒否されているなど、設定レベルで矛盾がある場合、最後の設定が考慮されます。

フィルタを無効にするには、コマンドを呼び出す際に空の配列を渡すか、配列の最後の要素で、filtersArr 配列に"*" を、allowDenyArr配列にTrueを渡します。

重要: WA SET URL FILTERSコマンドで設定されたフィルタが、WA SET EXTERNAL LINKS FILTERSコマンドよりも前に評価されます。つまりURLがWA SET URL FILTERSコマンドフィルタの設定により拒否されると、WA SET EXTERNAL LINKS FILTERSコマンドで受け入れていても、そのURL をブラウザで開くことはできません (例2 参照)。

例題 1

この例はサイトを外部ブラウザで開きます:

```
ARRAY STRING(0;$filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*www.google.*") `Select "google"
APPEND TO ARRAY($AllowDeny;False)
 `False: このリンクは外部ブラウザで開かれる
APPEND TO ARRAY($filters;"*www.apple.*")
APPEND TO ARRAY($AllowDeny;False)
 `False: このリンクは外部ブラウザで開かれる
WA SET EXTERNAL LINKS FILTERS(MyWArea;$filters;$AllowDeny)
```

例題 2

この例はサイトと外部リンク両方のフィルタを使用します:

```
ARRAY STRING(0;$filters;0)
ARRAY BOOLEAN($AllowDeny;0)
APPEND TO ARRAY($filters;"*www.google.*") `Select "google"
APPEND TO ARRAY($AllowDeny;False) `Deny this link
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)

ARRAY STRING(0;$filters;0)
ARRAY BOOLEAN($AllowDeny;0)
APPEND TO ARRAY($filters;"*www.google.*") `Select "google"
APPEND TO ARRAY($AllowDeny;False)
 `False: このリンクは外部ブラウザで開かれるべきだが、この設定は
 `URL フィルタによりブロックされ、無効となる。
WA SET EXTERNAL LINKS FILTERS(MyWArea;$filters;$AllowDeny)
```

WA SET PAGE CONTENT

WA SET PAGE CONTENT ({* ;} object ; content ; baseURL)

引数	型		説明
*	演算子	➡	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	➡	オブジェクト名 (* 指定時) または 変数 (* 省略時)
content	文字	➡	HTMLソースコード
baseURL	文字	➡	相対参照に使用するURL (Mac OS)

説明

The WA SET PAGE CONTENT コマンドは、* と object 引数で指定したWebエリアに表示されているページを、content引数で渡されたHTMLコードで置き換えます。

Mac OSでは、baseURL引数を使用して、ページ中に存在する相対パスの前に追加するベースURLを指定できます。
Windows では、この引数は効果がなく、ベースURLは指定されません。このプラットフォームでは相対参照を使用できません。

Note: Windows では、このコマンドが呼ばれる前に、ページが既にWeb エリアにロードされていなければなりません。必要であれば"about:blank"URL を使用してブランクページをロードできます。

例題

"Hello world!" を表示して、ベースURL を"file:///" にします (Mac OS のみ):

```
WA SET PAGE CONTENT(MyWArea;"<html><body><h1>Hello World!</h1></body></html>";"file:///")
```

WA SET PAGE TEXT LARGER

WA SET PAGE TEXT LARGER ({* ;} object)

引数	型		説明
*	演算子	⇒	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または 変数 (* 省略時)

説明

WA SET PAGE TEXT LARGER コマンドは、* と object 引数で指定したWebエリアに表示されているテキストのサイズを大きくします。

Mac OS では、このコマンドのスコープは4D セッションとなります。このコマンドにより実行される設定は、4D アプリケーション終了後は引き継がれません。
Windows では、このコマンドのスコープはグローバルです。4D アプリケーション終了後も設定が引き継がれます。

WA SET PAGE TEXT SMALLER

WA SET PAGE TEXT SMALLER ({* ;} object)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)

説明

WA SET PAGE TEXT SMALLER コマンドは、* と object 引数で指定したWebエリアに表示されているテキストのサイズを小さくします。

Mac OS では、このコマンドのスコープは4D セッションとなります。このコマンドにより実行される設定は、4D アプリケーション終了後は引き継がれません。
Windows では、このコマンドのスコープはグローバルです。4D アプリケーション終了後も設定が引き継がれます。

WA SET PREFERENCE

WA SET PREFERENCE ({* ;} object ; selector ; value)

引数	型		説明
*	演算子	→	指定した場合、オブジェクトがオブジェクトの名前 (文字列) 省略した場合、オブジェクトは変数
object	フォームオブジェクト	→	オブジェクトの名前 (* を指定した場合) または、変数 (* を省略した場合)
selector	倍長整数	→	修正される環境設定
value	ブール	→	環境設定の値 (True = 許可, False = 不許可)

説明

WA SET PREFERENCE コマンドを使用して、引数 * と object によって指定されたWebエリアに対して、さまざまな環境設定を行います。

引数selectorに修正する環境設定を渡し、引数valueにその環境設定に割り当てられる値を渡します。引数selector には、Web Area テーマにある以下の定数の1つを渡します。

定数	型	値	コメント
WA enable contextual menu	倍長整数	4	Webエリア内で標準のコンテキストメニューの表示を許可する
WA enable Java applets	倍長整数	1	Webエリア内でJava appletの実行を許可する
WA enable JavaScript	倍長整数	2	Webエリア内でJavaScriptコードの実行を許可する
WA enable plugins	倍長整数	3	Webエリア内でプラグインのインストールを許可する
WA enable Web inspector	倍長整数	100	Web エリア内でインスペクターの表示を許可する

各環境設定を起動するには値に True を渡し、無効にするにはFalse を渡します。

WA SET URL FILTERS

WA SET URL FILTERS ({* ;} object ; filtersArr ; allowDenyArr)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
filtersArr	文字配列	→	フィルタ配列
allowDenyArr	ブール配列	→	許可-拒否配列

説明

WA SET URL FILTERS コマンドは、* と object 引数で指定したWebエリアで、1 つ以上のフィルタを設定するために使用します。

ユーザからリクエストされたページをロードする前に、4D はフィルタのリストを照会し、ターゲットのURL に接続が許可されているかどうかを調べます。
URLの判定はfiltersArr とallowDenyArr 配列の内容に基づき行われます。

リクエストされたURL が許可されない場合、ページはロードされず、On URL Filtering フォームイベントが生成されます。

filtersArr と allowDenyArr 配列は同期されていなければなりません。

- filtersArr 配列のそれぞれの要素には、フィルタするURL が含まれます。1 つ以上の文字を表すワイルドカードとして * を使用できます。
- allowDenyArr 配列のそれぞれ対応する要素には、URL を許可 (True) するか拒否 (False) するかを示すブール値が含まれます。

同じURL が許可および拒否されているなど、設定レベルで矛盾がある場合、最後の設定が考慮されます。

フィルタを無効にするには、コマンドを呼び出す際に空の配列を渡すか、配列の最後の要素で、filtersArr 配列に"*" を、allowDenyArr 配列
にTrueを渡します。

コマンドが実行されると、フィルタはWeb エリアのプロパティとなります。filtersArr とallowDenyArr が削除されたり初期化されたりしても、コマンドが再実行されるまでフィルタは有効です。エリアで有効になっているフィルタを取得するには、WA GET URL FILTERS コマンドを使用しなければなりません。

重要: このコマンドによって実行されるフィルタはWeb エリアに関連付けられた"URL" 変数にのみ適用されます (変数は通常入力可で、フォームに表示されます)。
フィルタはWA OPEN URL コマンドや他のナビゲーションコマンドには適用されません。

例題 1

.org, .net そして .fr Web サイトへのアクセスを禁止したい場合:

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*.org")
APPEND TO ARRAY($AllowDeny;False)
APPEND TO ARRAY($filters;"*.net")
APPEND TO ARRAY($AllowDeny;False)
APPEND TO ARRAY($filters;"*.fr")
APPEND TO ARRAY($AllowDeny;False)
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
```

例題 2

日本のサイト以外へのアクセスを禁止したい場合(.jp):

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*") `Select all
APPEND TO ARRAY($AllowDeny;False) `Deny all
APPEND TO ARRAY($filters;"www.*.jp") `Select *.jp
APPEND TO ARRAY($AllowDeny;True) `Allow
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
```

例題 3

4D のWeb サイトにのみアクセスを許可する場合 (.com, .fr, .es, etc.):

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*") `Select all
APPEND TO ARRAY($AllowDeny;False) `Deny all
APPEND TO ARRAY($filters;"www.4D.*") `Select 4d.fr, 4d.com...
APPEND TO ARRAY($AllowDeny;True) `Allow
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
```

例題 4

ローカルのドキュメントにのみアクセスを許可 (C://doc フォルダ内):

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*") `Select all
APPEND TO ARRAY($AllowDeny;False) `Deny all
APPEND TO ARRAY($filters;"file://C:/doc/*")
 `Select the path file:// allowed
APPEND TO ARRAY($AllowDeny;True) `Allow
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
```

例題 5

特定のキーワードを含むサイトを除いて許可する場合:

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*")
APPEND TO ARRAY($AllowDeny;True) `Allow all
APPEND TO ARRAY($filters;"*elcaro*") `Deny all that contain elcaro
APPEND TO ARRAY($AllowDeny;False)
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
```

例題 6

特定のIP アドレスへのアクセスを拒否する場合:

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)
APPEND TO ARRAY($filters;"*") `Select all

APPEND TO ARRAY($AllowDeny;True) `Allow all
APPEND TO ARRAY($filters;86.83.*")  `Select IP addresses beginning with 86.83.
APPEND TO ARRAY($AllowDeny;False) `Deny
APPEND TO ARRAY($filters;86.1*")  `Select IP addresses beginning with 86.1 (86.10, 86.135 etc.)
APPEND TO ARRAY($AllowDeny;False) `Deny
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
 `(Note that the IP address of a domain may vary).
```

WA STOP LOADING URL

WA STOP LOADING URL ({* ;} object)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)

説明

WA STOP LOADING URL コマンドは、* と object 引数で指定したWebエリアの現在のURL のリソース読み込みを停止します。

Webサーバ

- Webサーバ概要
- Webサーバ設定と接続管理
- Support of IPv6
- 接続セキュリティ
- On Web Authenticationデータベースメソッド
- On Web Connectionデータベースメソッド
- On Web Session Suspend データベースメソッド
- Webセッション管理
- 4DオブジェクトをHTMLオブジェクトにバインドする
- URLとフォームアクション
- 4D HTMLタグ
- Webサーバー設定
- Webサイトに関する情報
- SSLプロトコルの使用
- XMLとWMLサポート
- WEB CLOSE SESSION
- WEB GET BODY PART
- WEB Get body part count
- WEB Get Current Session ID
- WEB GET HTTP BODY
- WEB GET HTTP HEADER
- WEB GET OPTION
- WEB GET SESSION EXPIRATION
- WEB GET STATISTICS
- WEB GET VARIABLES
- WEB Is secured connection
- WEB SEND BLOB
- WEB SEND FILE
- WEB SEND HTTP REDIRECT
- WEB SEND RAW DATA
- WEB SEND TEXT Updated 14.0
- WEB SET HOME PAGE
- WEB SET HTTP HEADER
- WEB SET OPTION
- WEB SET ROOT FOLDER
- WEB START SERVER
- WEB STOP SERVER
- WEB Validate digest
- _o_SET CGI EXECUTABLE
- _o_SET WEB DISPLAY LIMITS
- _o_SET WEB TIMEOUT
- _o_Web Context

Webサーバ概要

4Dのローカルモード、リモートモード、および4D ServerにはWebサーバーエンジンがあります。このWebサーバーエンジンを使用して、4DデータベースまたはあらゆるタイプのHTMLページを Web上に公開することができます。4DのWebサーバーエンジンには以下のような主な特徴があります。

- **簡単に公開**
  好きな時にデータベースの公開をWeb上で開始または停止することができます。これを実行するには、メニューコマンドを選択、またはランゲージコマンドを実行するだけです。

- **専用データベースメソッド**
  On Web Authenticationデータベースメソッド と On Web Connectionデータベースメソッドは、Webサーバーにおいてリクエストのエントリポイントになります。これらを使用して、あらゆるタイプのリクエストを評価して適切な処理を行うことができます。

- **特殊タグとURLの使用**
  4DのWebサーバーはユーザアクションとの相互作用を可能にする多くのメカニズムを提供しています。以下のような特別な機能があります。
  - Webページに含める特別なタグ。これはWebページがブラウザーに送られる際、Webサーバーによって処理を開始するためのものです。
  - あらゆるアクションを実行するために4Dの呼び出しを可能にする特殊なURL。
  - これらのURLをフォームアクションとして使用して、ユーザがHTMLフォームを投稿した際、処理を始動させます。

- **ユーザーセッションの管理**
  4D Webサーバーにはcookieを使用する、完全に自動化されたWebセッション (ユーザーセッション) 管理機能があります。
- **アクセスセキュリティ**
  複数の自動設定オプションを使用すると、Webブラウザーに特別なアクセス権を与えたり、4Dに統合されたパスワードシステムを使用することが可能になります。"一般Webユーザ" を定義して、データベース内でのアクセス管理を簡略化できます。

  On Web Authenticationデータベースメソッドを用て、Webサーバーがリクエストを処理する前に、そのリクエストを評価できます。更にデフォルトのHTMLルートフォルダーを定義する機能はディスク上のファイルへのアクセスを制限します。

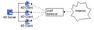
  最後に、Web経由での実行を許可するプロジェクトメソッドは個別に指定しなくてはなりません。

- **SSL接続**
  4DのWebサーバーは、SSLプロトコル (Secured Socket Layer) を通じて、ブラウザーと保護モードで通信できます。ほとんどのWebブラウザーと互換性のあるこのプロトコルは送信者と受信者を認証し、交換された情報の機密性と整合性を保証します。

- **インターネットフォーマットの拡張サポート**
  4DのWebサーバーは、HTTP/1.1 と互換性があり、XMLドキュメントとWML (Wireless Markup Language) テクノロジーをサポートしています。
  また4D Webサーバーはzgip圧縮もサポートしています。Webサーバーとクライアント間でネゴシエーションが行われた後、パフォーマンス向上のため、可能であればデータの圧縮が行われます。
- **データベースを同時に操作する**
  - **4DのローカルモードとWeb**
    4Dのローカルモードを使用して4DのデータベースをWeb上で公開する場合、以下のことを同時に行えます。
    - 4Dでデータベースをローカルで使用する。
    - Webブラウザーを用いてデータベースへ接続する。
  - **4DサーバーとWeb**
    4Dサーバーを使用して4DのデータベースをWeb上で公開する場合、以下を使用して4Dのデータベースへの接続とその処理を同時に行えます。
    - 4Dリモートワークステーション
    - Webブラウザー
  - **4DクライアントとWeb**
    Web上で4Dのデータベースが4Dクライアントによって公開されている場合、4Dのデータベースへ接続し、以下を通じて同時に使用します。
    - 4Dリモートマシン経由
    - Web ブラウザー経由。この場合、データベースが4D Serverで公開されていると、Webブラウザーは4Dクライアントマシン経由または4D Server経由でその公開されているデータベースへ接続できます。更に、異なるデータへのアクセスモードが利用できます (パブリック、アドミニストレーションなど) 。

    4D Webサーバーの基本的なメカニズムはリモートモードの4Dでも同様に使用されています。コマンドがローカルモードの4D、4D Server、あるいはリモートモードの4Dで実行されようとも、ランゲージコマンドの操作は通常同じです。ポイントはコマンドはそれが実行されるマシンのWebサイトに適用されるということです。Execute on server / EXECUTE ON CLIENTコマンドを使用してこれを管理する必要があります。

- **4Dクライアントでのロードバランス**:
  4Dマシンのリモートモードは Webサーバーとして使用でき、ロードバランサーを用いてダイナミックなWebサーバーを設定することができます。これにより、特に以下を含む、広範囲に及ぶ開発が可能になります。
  - すべての4DのWebサーバーにインストールされているWebサイトのミラーを利用して、4DのWebサーバーのパフォーマンスを最適化するためにロードバランスシステムを設定することができます。(ハードウェアーまたはソフトウェアーによる) ロードバランサーがカレントロードに基づいて、クライアントマシンにリクエストを送ります。

  - フォールトトレランスのWebサーバー設定。4DのWebサイトは2つ以上の4Dクライアントマシンでミラーされています。1つの4D Webサーバーが故障しても、他のWebサーバーが機能します。
  - 例えばリクエスト元に応じて同じデータで異なる表示を作成する。企業のネットワーク内で、保護された4Dクライアントマシン上のWebサーバーはイントラネットのリクエストを供給することができます。そしてファイアーウォールの向こう側に設置されているほかのクライアントマシン上のWebサーバーはインターネットからのリクエストに対応します。
  - 異なる4Dクライアントマシン上のWebサーバー間でタスクを配信します。4DのWebサーバーの1つはSOAPリクエストを管理し、もう一方のWebサーバーは標準のリクエストを処理します。

Webサーバ設定と接続管理

4D と 4D Serverは透過的また動的に、あなたのデータベースをWebに配信するWebサーバー (HTTPサーバー) を含んでいます。
この節ではWebサーバーの起動とブラウザーからの接続に必要なステップ、および接続管理の処理について説明します。

4DデータベースをWebに公開する条件

4Dや4D ServerのHTTPサーバーを起動するためには、以下が必要です:

- "4D Web Application" ライセンス。詳細は4Dのインストールガイドを参照してください。
- Web接続はTCP/IPを使用してネットワーク経由で行われます。そのため:
  - マシンにTCP/IPがインストールされていて、正しく設定されていなければなりません。詳細はコンピューターまたはOSのマニュアルを参照してください。
  - ネットワーク接続にSSLを使用したい場合は、必要なコンポーネントが正しくインストールされているか確認します (SSLプロトコルの使用参照)。

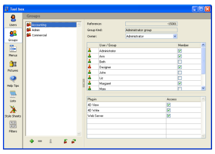
以上の点をすべてチェックした後に、4D内でWebサーバーを開始します。この点についてはこの節の後で説明します。

公開認証 (リモートモードの4D)

デフォルトでは、すべての4DクライアントマシンからデータベースをWebに公開できます。しかし個々の4Dリモートマシンごとに Web公開を行うかどうかを、4Dのパスワードシステムを使用してコントロールできます。
実際、4D Webライセンスは4D Serverによりプラグインライセンスとして扱われます。すなわちプラグインと同じ方法で、Webサーバーライセンスを使用する権限を特定のユーザーグループに制限しなければなりません。

これを行うには、4Dでツールボックスの**グループ**ページを表示します (これらのパラメーターを変更する権限を持っていなければなりません)。

左のリストからグループを選択し、プラグインアクセス権エリアの**4D Client Web Server**の隣の**アクセス**オプションをチェックします:


Tool box
Users
Groups
Menus
Pictures
Help Tips
Lists
Style Sheets
Filters
Groups
Accounting
Admin
Commercial

Reference: -15001
Group Kind: Administrator group
Owner: Administrator

	User / Group	Member
	Administrator	☑
	Ann	☑
	Beth	☐
	Designer	☑
	John	☐
	Liz	☐
	Margaret	☑
	Mary	☐

Plugin	Access
4D View	☑
4D Write	☑
Web Server	☑

上記の作業により、許可されたグループのユーザーだけがWebサーバーとして4Dマシンを公開できます。

Mac OS XでのHelperTool

Mac OS Xでは、0~1023のポートを利用するために特別な権限が必要です。これらのポートを利用可能にするために、4DはHelperToolというユーティリティプログラムを提供しています。このプログラムがインストールされると、プログラムは適切な権限を取得し、Webポートを開くための作業を自動で行います。このメカニズムは4D (すべてのモード)、4D Server、そして4D Volume Desktopで動作します。

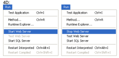
HelperToolアプリケーションは4Dソフトウェアに含まれます。インストールは最初に1023未満のポートを開く際自動で行われます。ツールがインストールされる旨通知され、管理者の名前とパスワードの入力を要求されます。この処理は一回だけ必要です。アプリケーションは"com.4D.HelperTool"と名称変更され、"/Library/PrivilegedHelperTools/"にインストールされます。一連の作業の後、4D Webサーバーは、使用する4Dのバージョンに関わらず、透過的に開始および停止できます。

4D Webサーバの開始

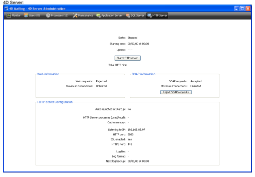
4D Webサーバーは3つの方法で起動できます:

- 4Dの**実行**メニューまたは4D ServerのHTTPサーバーページ (**HTTPサーバー開始**ボタン)を使用する。これらのコマンドを使用していつでもWebサーバーを開始したり停止したりできます:

4D:

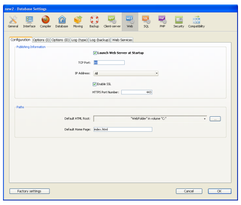

Run
Test Application Ctrl+I
Method... Ctrl+R
Runtime Explorer...
Start Web Server
Test Web Server
Start SQL Server
Restart Interpreted Ctrl+Alt+I
Restart Compiled Ctrl+Shift+I
Run
Test Application Ctrl+I
Method... Ctrl+R
Runtime Explorer...
Stop Web Server
Test Web Server
Start SQL Server
Restart Interpreted Ctrl+Alt+I
Restart Compiled Ctrl+Shift+I

4D Server:


4D Mailing - 4D Server Administration
Monitor
Users (0)
Processes (11)
Maintenance
Application Server
SQL Server
HTTP Server
Start HTTP server
Web information
SOAP information
Reject SOAP requests
HTTP server Configuration

State: Stopped
Starting time: 00/00/00 at 00:00
Uptime: --:--
Total HTTP hits:
Web requests: Rejected
Maximum Connections: Unlimited
SOAP requests: Accepted
Maximum Connections: Unlimited
Auto-launched at startup: No
HTTP Server processes (used/total): -
Cache memory: -
Listening to IP: 192.168.88.97
HTTP port: 8080
SSL enabled: Yes
HTTPS Port: 443
Log file: -
Log format: -
Next log backup: 00/00/00 at 00:00

- 4Dアプリケーションが開かれるときに自動で開始する。このようにするにはデータベース設定の、Webテーマの**設定**ページを表示します:


new2 - Database Settings
General
Interface
Compiler
Database
Moving
Backup
Client-server
Web
SQL
PHP
Security
Compatibility
Configuration
Options (I)
Options (II)
Log (type)
Log (backup)
Web Services
Publishing Information
Paths
Factory settings
Cancel
OK

☑ Launch Web Server at Startup
TCP Port: 80
IP Address: All
☑ Enable SSL
HTTPS Port Number: 443
Default HTML Root: "WebFolder" in volume "C:"
Default Home Page: index.html

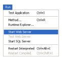
公開情報エリアで、**開始時にWebサーバーを起動**チェックボックスにチェックをして**OK**をクリックします。これを行えば、4Dや4D Serverを起動するたびにWebが自動で公開されます。

- WEB START SERVERコマンドを呼び出してプログラムで開始する。

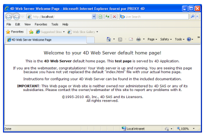
Tip: データベースをWebに公開するために、4Dを終了して再起動する必要はありません。必要な時にWebサーバーを停止し、また開始することができます。これを行うには**実行**メニューを使用する、**HTTPサーバー開始**ボタンを使用する、またはWEB START SERVER とWEB STOP SERVERコマンドを使用します。

Webサーバのテスト

Webサーバテスト メニューコマンドを使用してビルトインのWebサーバが正しく実行されているか確認できます (4Dのみ)。このメニューはWebサーバが実行されているときに**実行**メニューからアクセスできます:


Run
Test Application Ctrl+I
Method... Ctrl+R
Runtime Explorer...
Start Web Server
Test Web Server
Start SQL Server
Restart Interpreted Ctrl+Alt+I
Restart Compiled Ctrl+Shift+I

このコマンドを選択すると、4Dアプリケーションが公開しているWebサイトのホームページが、デフォルトWebブラウザに表示されます:


4D Web Server Welcome Page - Microsoft Internet Explorer fourni par PROXY 4D
http://localhost/
Live Search
File
Edit
View
Favorites
Tools
Help
Favorites
Suggested Sites
Web Slice Gallery
4D Web Server Welcome Page
Page
Safety
Tools
Done
Local intranet
100%

Welcome to your 4D Web Server default home page!

This is the **4D Web Server** default home page. This **test page** is served by 4D Application.

If you are the webmaster, congratulations! Your Web server is up and running. You are seeing this page because you have not yet replaced the default "index.html" file with your actual home page.

Instructions for configuring your 4D Web Server can be found in the included documentation.

IMPORTANT: This Web page or Web site is neither owned nor administered by 4D SAS or any of its subsidiaries. Please contact the owner/webmaster of this site to report any problems with it.

©1995-2010 4D, Inc., 4D SAS and its Licensors.
All rights reserved.

このコマンドでWebサーバが起動され、ホームページが表示可能かなどを検証できます。ページは、Webブラウザが実行されているマシンのIPアドレスを指定する標準のショートカットである、ローカルホストのURLを使用して呼び出されます。コマンドはデータベース設定で設定されたWeb公開ポート番号を考慮に入れます。

Web上に公開された4Dデータベースに接続する

4DデータベースをWebに公開した後、Webブラウザーから接続ができるようになります。これを行うには:

- Webサイトが登録名を持つ場合 (例 "www.flowersforever.com")、ブラウザーのアドレス欄にその名前を入力し、**Enter**キーを押します。
- Webサイトに登録名がない場合、マシンのIPアドレス (例 123.4.567.89) をブラウザーのアドレス欄に入力し、**Enter**キーを押します。

この時点で、ブラウザーにはWebサイトのホームページが表示されるはずです。データベースを標準の設定で公開しているなら、4D Webサーバーのデフォルトホームページが表示されます。

以下のような状況になるかもしれません:

1. 接続に失敗し、以下のようなメッセージが表示される "...サーバーが接続を許可していないか、応答できないようです..."。この場合、以下の点をチェックします:
   - 入力したWebサーバーアドレスまたはIPアドレスが正しいか確認する。
   - 4D/4D Serverが起動され、Webサーバーが開始されているか確認する。
   - WebのTCPポートがデフォルトのWeb TCPポート以外に設定されていないか確認する (状況4参照)。
   - サーバーマシンとブラウザーマシン両方でTCP/IPが正しく設定されているか確認する。両マシンは同じサブネットに存在するか、ルーターが正しく設定されていなければなりません。
   - ハードウェアーの接続を確認する。
   - ローカルサイトではなく、インターネットやイントラネット上の誰かが提供しているWebデータベースに接続しようとしている場合、メッセージは実際の状況を表しているかもしれません。この場合、後ほど試すか、Webプロバイダーに連絡します。
2. 接続したがHTTP 404 "ファイルが見つかりません"エラーを受け取った。これはホームページが提供されていないことを意味します。この場合、データベース設定で定義した場所に、ホームページが存在するか (Webサーバー設定ページ参照) 、またはWEB SET HOME PAGEコマンドを使用しているか確認してください。
3. 接続はしたが、期待したWebページが表示されなかった。これは複数のWebサーバが同じマシン上で動作しているときに発生します。例えば:
   - すでにWebサーバーが起動されているWindows上で1つの4D Webデータベースを実行した。
   - 同じマシン上で複数の4D Webデータベースを実行した。

このような状況では、4D Webデータベースのポート番号を変更する必要があります。これを行うにはWebサーバー設定を参照してください。

注: データベースがパスワードシステムで保護されている場合、有効なユーザー名とパスワードを入力しなければならないかもしれません (詳細は接続セキュリティを参照してください)。

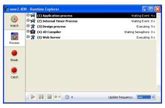
Web プロセス管理

複数の4DプロセスがデータベースのWeb公開とブラウザとの接続をサポートします。この段落はこれらのプロセスとその特徴について説明します。

Web サーバプロセス

Web サーバープロセスはデータベースがWebサイトとして公開されると実行されます。

ランタイムエクスプローラの**プロセス**ページには、実行されているWebサーバプロセスが表示されます。


new2.4DB - Runtime Explorer
Watch
Process
Break
Catch
Update frequency:
Low

(1) Application process Waiting Event 4 s
(2) Internal Timer Process Waiting Event 0 s
(3) Design process Executing 9 s
(4) 4D Compiler Waiting Semaphore 0 s
(5) Web Server Executing 0 s

これは4Dのカーネルプロセスであり、**アボート**ボタンでアボートさせることはできません。またCALL PROCESS等を使用してプロセス間通信を行うこともできません。Webサーバープロセスはまったくユーザーインタフェースコンポーネント (ウィンドウ、メニューなど) を持っていないことに注意してください。

Webサーバープロセスは以下の方法で開始できます:

- 4D Serverの"HTTPサーバー"ページで、**HTTPサーバー開始** ボタンをクリックするか、4Dの**実行**メニューから**Webサーバー開始**を選択する。
- WEB START SERVERを呼び出す。
- **開始時にWebサーバーを起動**が環境設定で設定されたデータベースを開く。

Webサーバープロセスは以下の方法で停止できます:

- 4D Serverの"HTTPサーバー"ページで、**HTTPサーバー停止** ボタンをクリックするか、4Dの**実行**メニューから**Webサーバー停止**を選択する。
- WEB STOP SERVERを呼び出す。
- データベースを終了する。

Web サーバープロセスの目的は、Web接続の試みを処理することだけです。Webサーバープロセスを開始することは実際のWeb接続を開くことを意味せず、単にWebユーザーがWeb接続を開始することを許可することを意味します。Webサーバープロセスの停止は、現在実行されているWeb接続プロセスを閉じることを意味せず、単にWebユーザーが新しいWeb接続を開始できなくなることを意味します。

Webサーバープロセスを停止するとき実行中のWeb接続プロセスがあると、これらのプロセスは通常通り実行が継続されます。
すなわち、Webサーバープロセスが完全に停止するには遅延時間が必要です。

Web プロセス

Webブラウザーがデータベースへの接続を試みるたびに、リクエストは以下の手順でWebサーバープロセスで処理されます:

- まずWebサーバープロセスは1つあるいは複数の一時的な4Dローカルプロセスを作成します。これは**Webプロセス**と呼ばれ、Webブラウザーとの接続を検証、管理します。
  この一時的なプロセスはすべてのHTTPリクエストを管理します。これは素早く実行され、アボートされるか遅延されます。非コンテキストモードでWebサーバの応答性を向上するために、4DはWebプロセスのプールを5秒遅延させ、将来のHTTPクエリの実行時に再利用します。この振る舞いはSET DATABASE PARAMETERコマンドで変更できます。
- Webプロセスはリクエストの処理をハンドルし、必要であればレスポンスをブラウザーに送信します。一時的なプロセスはアボートまたは遅延されます (上記参照)。

Support of IPv6

v14以降、4D は IPv6 アドレス記法をサポートするようになりました。この新機能は4D v14の統合サーバー(下記)に関係します:

- Web サーバーと SOAP サーバー
- SQL サーバー

注: IPv6 についての詳細な情報は、以下の詳細を参照して下さい: RFC 2460

IPv6 のサポートは 4D ユーザーや 4D デベロッパが気づくことはありません。プログラムはサーバーの「IPアドレス」が**すべて**に設定されていればIPv6接続でもIPv4接続でも無差別に受け入れます(HTTPリクエストを受け付けるIPアドレスの指定 (HTTP server) と SQLサーバの公開設定 (SQL server) を参照して下さい)。

ただし、以下の点に気を付ける必要があります:

- **ポート番号の表記**
  IPv6 記法はコロン (:) を使用するので、ポート番号を追加するときには混乱を招く恐れがあることに注意して下さい。例えば以下の様な場合です:
  ```
  2001:0DB8::85a3:0:ac1f:8001 // IPv6 アドレス
  2001:0DB8::85a3:0:ac1f:8001:8081 // ポート 8081指定の IPv6 アドレス
  ```
  混乱を避けるため、IPv6 アドレスをポート番号と併用する際には、以下の様に [] でアドレスを囲う記法が推奨されます:
  ```
  [2001:0DB8::85a3:0:ac1f:8001]:8081 //ポート 8081指定の IPv6 アドレス
  ```
- **TCPポートが使用されている際の警告は出ません**
  これまでのバージョンの4Dと異なり、4D v14ではサーバーが応答するIPアドレスが「全て」に設定されていた場合には、TCPポートが他のアプリケーションで使用されていてもそれはサーバー起動時に表示されなくなりました。IPv6アドレスがあいているため、この場合4D Serverはどのようなエラーも検知しなくなりました。しかしながら、マシンのIPv4アドレスを使用、またはローカルアドレス127.0.0.1.を使用してアクセスすることは不可能になりました。
  4D serverが定義されたポートで反応していないようであれば、サーバーマシンで [::1] のアドレスを試してみてください(IPv6における 127.0.0.1 と同義です。他のポート番号をテストするためには *portNum* を追加して下さい)。4Dが応答するようであれば、他のアプリケーションがポートをIPv4で使用している可能性が高いです。

接続セキュリティ

4D Webサーバーのセキュリティは以下の要素に基づきます:

- Webパスワード管理システム (BASIC モードとDIGESTモード) とOn Web Authenticationデータベースメソッドの組み合わせ
- 一般Webユーザーの定義
- デフォルトHTMLルートフォルダーの定義
- プロジェクトメソッドごとの"4DタグおよびURL (4DACTION) で利用可"プロパティの定義
- HTTPを使用した同期リクエストのサポートに関する設定

注: 接続自身のセキュリティはSSLプロトコルで管理できます。詳細な情報はSSLプロトコルの使用を参照してください。

Webアクセスのパスワード管理システム

BASICモードとDIGESTモード

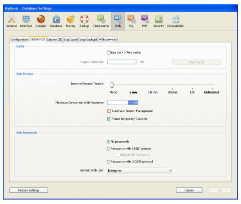
データベース設定で、Webサーバーに適用するアクセスコントロールシステムを設定できます。2つの認証モード、BASICモードとDIGESTモードが提供されています。認証モードはユーザー名とパスワードに関する情報の取得と処理方法に関連します。

- BASICモードでは、ユーザーが入力したユーザー名とパスワードが暗号化されずにHTTPリクエストに含められて送信されます。この場合情報は第三者に盗聴・使用される可能性があるので、トータルなセキュリティは確保されません。
- DIGESTモードはより高いセキュリティレベルを提供します。認証情報は復号が困難な一方向ハッシュを使用して処理されます。

ユーザーにとり、いずれの認証モードを使用するかは透過的です。

注:
互換性の理由から、バージョン11に変換された4Dデータベースでは、BASIC認証モードがデフォルトで使用 されます ("パスワードを使用" オプションが以前のバージョンでチェックされていれば)。Digestモードを使用するには明示的に指定しなければなりません。
Digest認証 はHTTP1.1の機能で、すべてのブラウザーでサポートされているわけではありません。例えばバージョン5.0以降のMicrosoft Internet Explorerがこのモードを受け入れます。Digestモードが有効な時に、この機能をサポートしないブラウザーがWebサーバにリクエストを送信すると、サーバーはリクエストを拒否し、エラーメッセージをブラウザーに返します。

データベース設定ダイアログボックスで、Webサーバーに適用するアクセスコントロールシステムを定義できます。これを行うには**Web**テーマの**オプション (I)**ページを表示します:


MyMusic - Database Settings
General
Interface
Compiler
Database
Moving
Backup
Client-server
Web
SQL
PHP
Security
Compatibility
Configuration
Options (I)
Options (II)
Log (type)
Log (backup)
Web Services
Cache
Web Process
Web Passwords
Clear Cache
None
5 mn
15 mn
30 mn
1 h
Unlimited
Factory settings
Cancel
OK

☐ Use the 4D Web cache
Pages Cache Size: 0 Kb
Inactive Process Timeout: None
Maximum Concurrent Web Processes: 32000
☐ Automatic Session Management
☑ Reuse Temporary Contexts
● No passwords
○ Passwords with BASIC protocol
☐ Include 4D Passwords
○ Passwords with DIGEST protocol
Generic Web User: Designer

"Webパスワード"エリアで、3つのオプションから選択できます:

- **パスワードなし**: Webサーバーへの接続に認証を実行しない。この場合:
  - On Web Authenticationデータベースメソッドが 存在すれば、それが実行され、$1と$2に加え、ブラウザーとサーバーのIPアドレス ($3と$4) だけが提供されます。ユーザー名とパスワード ($5と$6) は空です。この場合、ブラウザーのIPアドレスやリクエストされたサーバーのIPアドレスを使用して、リクエストをフィルターできます。
  - On Web Authenticationデータベースメソッドが存在しない場合、接続は自動で受け入れられます。
- **BASIC認証のパスワード**: BASICモードの標準認証です。ユーザーがサーバーに接続するとダイアログボックスがブラウザー上に表示され、ユーザー名とパスワードの入力を求められます。これら2つの値は他の接続パラメーター (IPアドレス、URI) とともにOn Web Authenticationデータベースメソッドに送信され、開発者はそれを処理できます。

  注: この場合On Web Authenticationデータベースメソッドが存在しないと、接続は拒否されます。

  このモードを有効にすると、**4Dパスワードを含む**オプションが選択できるようになり、カスタムパスワードシステムの代わりに、あるいは追加として使用できます。
- **DIGEST認証のパスワード**: DIGESTモードの認証。BASICモードのように、ユーザーは接続時にユーザー名とパスワードを入力します。これら2つの値は暗号化されて、他の接続パラメーターとともにOn Web Authenticationデータベースメソッドに送信されます。この場合WEB Validate digestコマンドを使用してユーザーを認証しなければなりません。

これらの設定を変更した場合は、Webサーバーを再起動しなければなりません。
4Dリモートモードのwebサーバーでは、すべての4Dリモートモードのマシンが同じユーザーテーブルを共有することを覚えておいてください。ユーザー/パスワードの検証は4D Serverアプリケーションで行われます。

BASICモード: パスワードとOn Web Authentication データベースメソッドの組み合わせ

BASICモードを使用する場合、4D Webサーバーへの接続をフィルターするシステムは、2つのパラメーターの組み合わせに基づきます:

- データベース設定環ダイアログボックスのWebパスワードオプション
- On Web Authenticationデータベースメソッドの存在

組み合わせは以下のとおり:

"BASIC認証のパスワード"が選択され、"4Dパスワードを含む"が選択されていない場合:

- On Web Authenticationデータベースメソッドが存在すれば、それが実行され、すべての引数が渡されます。そしてユーザー名とパスワード、さらにブラウザーとWebサーバーのIPアドレスを使用して、より精密に接続をフィルターできます。
- On Web Authenticationデータベースメソッドが存在しない場合、接続は自動で拒否され、認証メソッドが存在しない旨のメッセージがブラウザーに送信されます。

注: ブラウザーから送信されたユーザー名が空の文字列で、かつOn Web Authenticationデータベースメソッドが存在しない場合、ブラウザーにパスワードダイアログボックスが送信されます。

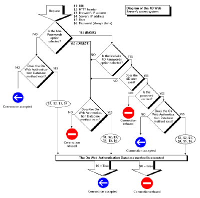
"BASIC認証のパスワード" と "4Dパスワードを含む" が選択されている場合:

- ブラウザーから送信されたユーザー名が4Dのユーザーテーブルに存在し、パスワードが正しい場合、接続は受け入れられます。パスワードが正しくなければ接続は拒否されます。
- ブラウザーから送信されたユーザー名が4Dに存在しない場合、2つの可能性があります:
  - On Web Authenticationデータベースメソッドが存在すれば、引数$1, $2, $3, $4, $5, $6に値が渡されます。ユーザー名とパスワード、さらにブラウザーとWebサーバーのIPアドレスを使用して、より精密に接続をフィルターできます。
  - On Web Authenticationデータベースメソッドが存在しない場合、接続は拒否されます。

DIGESTモード

BASICモードと異なり、DIGESTモードは標準の4Dパスワードシステムと互換がありません。4DパスワードをDigest Web認証に使用できません。このモードが選択されると"4Dパスワードを含む"オプションは選択不可となります。Webユーザーの認証は、(テーブル等を使用した) カスタマイズされた方法で管理しなければなりません。
DIGESTモードが有効の時、On Web Authenticationデータベースメソッドの$6 引数 (パスワード) は常に空の文字列が渡されます。実際このモードを使用するとき、この情報はネットワークからクリアテキストでは渡されません。この場合接続リクエストはWEB Validate digestコマンドを使用して検証しなければなりません。

4D Webサーバーのアクセスシステムの処理を以下に図示します:


Request
$1: URL
$2: HTTP header
$3: Browser's IP address
$4: Server's IP address
$5: User
$6: Password (always blank)
Diagram of the 4D Web Server's access system
Is the Use Passwords option selected?
NO
YES (BASIC)
YES (DIGEST)
Is the Include 4D Passwords option selected?
NO
YES
Does the 4D user exist?
NO
YES
Is the password correct?
NO
YES
Does the On Web Authentication Database method exist?
NO
YES
Connection accepted
Connection refused
$1, $2, $3, $4
$1, $2, $3, $4, $5, $6
The On Web Authentication Database method is executed
$0 = True
$0 = False
Connection accepted
Connection refused

robotsについて (セキュリティメモ)

特定のクローラー (クエリエンジン, スパイダー...) はWebサーバーやスタティックページをクロールします。クローラーにサイトへのアクセスをさせたくない場合、アクセスを許可したくないURLを指定できます。
これを行うには、ROBOTS.TXTファイルをサーバーのルートに置きます。このファイルの内容は以下の構造になっていなければなりません:

```
User-Agent: <name>
Disallow: <URL> または <beginning of the URL>
```

例題:

```
User-Agent: *
Disallow: /4D
Disallow: /%23%23
Disallow: /GIFS/
```

"User-Agent: *" は、すべてのクローラーが対象であることを示します。
"Disallow: /4D" は、クローラーに /4D から始まるURLへのアクセスを許可しないことを通知します。
"Disallow: /%23%23" は、クローラーに /%23%23 から始まるURLへのアクセスを許可しないことを通知します。
"Disallow: /GIFS/" は、クローラーに /GIFS/ フォルダーおよびそのサブフォルダーへのアクセスを許可しないことを通知します。

他の例題:

```
User-Agent: *
Disallow: /
```

この場合クローラーにサイト全体へのアクセスを許可しないことを通知します。

注: この指定に従うかどうかはクローラーの実装次第です。アクセスを拒否できることを保証するものではありません。

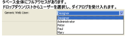
一般Webユーザー

事前に4Dパスワードテーブルに登録したユーザーを、"一般Webユーザー"として定義できます。この場合サーバーに接続するブラウザーは、それぞれこの一般ユーザーに割り当てられたアクセス認証や制限を使用できます。これによりデータベースの異なる部分へのブラウザーによるアクセスを簡単にコントロールできます。

注: ブラウザーからメソッドやフォームなどアプリケーションの様々なパーツへのアクセスを制限することを可能にするこのオプションと、パスワードシステムとOn Web Authenticationデータベースメソッドで管理するWebサーバー接続コントロールシステムとを、混同しないでください。

一般Webユーザーを定義するには:

1. デザインモードでツールボックスのユーザーエディターを開き、最低1人のユーザーを作成する。ユーザーにパスワードを設定することもできます。
2. 他の4Dエディターを使用して、このユーザーに必要なアクセス認証あるいは制限を設定します。
3. データベース設定ダイアログで、**Web**テーマの**オプション (I)**ページを開く。
   "Webパスワード"エリアに**一般Webユーザー** ドロップダウンリストがあります。デフォルトで一般WebユーザーはDesignerであり、ブラウザーはデータベース全体にフルアクセスがあります。
4. ドロップダウンリストからユーザーを選択し、ダイアログを受け入れます。



Generic Web User: Designer
Designer
Administrator
Peter
Paul
Mary

データベースへの接続が認証されたすべてのWebブラウザーは、一般Webユーザーに割り当てられたアクセス認証と制限を使用します (BASICモードと"4Dパスワードを含む"オプションがチェックされ、接続したユーザーが4Dパスワードテーブルに存在しない場合を除く、以下参照)。

BASIC認証との相互作用

"BASIC認証のパスワード"は一般Webユーザーがどのように動作するかには影響を与えません。このオプションのステータスにかかわらず、データベースに接続が許可されたすべてのWebブラウザーに、"一般Webユーザー"に割り当てられたアクセス認証と制限が適用されます。

しかし"4Dパスワードを含む"オプションが選択されていると、2つの可能性が発生します:

- ユーザー名とパスワードが4Dパスワードテーブルに存在しない場合。この場合接続がOn Web Authenticationデータベースメソッドで受け入れられると、一般Webユーザーのアクセス権がブラウザに適用されます。
- ユーザー名とパスワードが4Dパスワードテーブルに存在する場合、"一般Webユーザー"は無視され、ユーザーは固有のアクセス権で接続します。

デフォルトHTMLルート

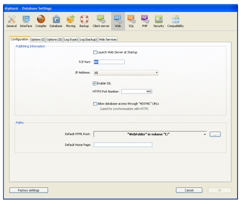
データベース設定のこのオプションを使用して、4Dがブラウザーに送信するスタティック/セミダイナミックなHTMLページ、ピクチャーなどを検索するフォルダーを指定できます。

さらにHTMLルートフォルダーは、Webサーバーのディスク上で、ファイルに対するアクセスができない階層を定義することにもなります。このアクセス制限はWebブラウザーのURLやWEB SEND FILEなどの4D Webサーバーコマンドに適用されます。ブラウザーからデータベースに送られたURLや4Dコマンドが、HTMLルートフォルダーよりも上の階層にアクセスしようとすると、サーバーエラーが返されます。

デフォルトで、4Dは**WebFolder**という名前のデフォルトHTMLフォルダーを定義します。Webサーバーの起動時にこのフォルダーが存在しなければ、HTMLルートフォルダーは物理的にディスク上に作成されます。
デフォルトの場所を保持すると、ルートフォルダーは以下の場所に作成されます:

- 4Dローカルモードと4D Serverでは、データベースストラクチャーファイルと同階層。
- 4Dリモートモードでは、4Dデータベースのローカルフォルダー (Get 4D folderコマンド参照)。

データベース設定で、HTMLルートフォルダーの名前と場所を変更できます (**Web**テーマの**設定**ページ):


MyMusic - Database Settings
General
Interface
Compiler
Database
Moving
Backup
Client-server
Web
SQL
PHP
Security
Compatibility
Configuration
Options (I)
Options (II)
Log (type)
Log (backup)
Web Services
Publishing Information
Paths
Factory settings
Cancel
OK

☐ Launch Web Server at Startup
TCP Port: 80
IP Address: All
☑ Enable SSL
HTTPS Port Number: 443
☐ Allow database access through "4DSYNC" URLs
(used for synchronization with HTTP)
Default HTML Root: "WebFolder" in volume "C:"
Default Home Page:

"デフォルトHTMLルート"入力エリアに、新しいフォルダーのパスを入力します。
このダイアログボックスに入力されるアクセスパスは相対パスです。起点はデータベースのストラクチャーファイルを含むフォルダ (4Dローカルモードまたは4D Server) 、または、4Dアプリケーションやソフトウェアーパッケージを含むフォルダーです (4Dリモートモード)。
データベースのマルチプラットフォーム互換性のため、4D Webサーバーはアクセスパスを定義するために特別な記法を使用します。シンタックスルールは以下のとおりです:

- フォルダーはスラッシュ ("/") で区切ります。
- アクセスパスはスラッシュ ("/") で終わってはいけません。
- フォルダー階層で1つ上にあがるには、フォルダー名の前にピリオドを2つ ".." 置きます。
- アクセスパスはスラッシュ ("/") で始まってはいけません (HTMLルートフォルダーをデータベースや4Dリモートモードのフォルダーにしたい場合を除く、以下参照)。

例えば、HTMLルートフォルダーを "4DDatabase" フォルダーのサブフォルダー "Web" にしたい場合、**4DDatabase/Web**と入力します。
HTMLルートフォルダーをデータベースフォルダーあるいは4Dリモートフォルダーにして、それより上階層へのアクセスを禁止したい場合、"/" を入力します。ボリュームへのアクセスを自由に行うためには、"デフォルトHTMLルート" エリアを空にします。

警告: データベース設定ダイアログボックスでデフォルトHTMLルートフォルダーを定義しないと、データベースのストラクチャーファイルあるいは4Dアプリケーションが含まれるフォルダーが使用されます。**この場合アクセス制限がありませんので注意してください** (ユーザーはすべてのボリュームにアクセスできます)。

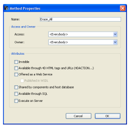
注:
HTMLルートフォルダーをデータベース設定ダイアログで更新すると、アクセスが制限されているファイルを格納しないようにするため、キャッシュがクリアされます。
WEB SET ROOT FOLDERコマンドを使用して動的にHTMLルートフォルダーを定義できます。この場合、更新はワークセッションすべてのカレントWebプロセスに適用されます。HTMLページのキャッシュはクリアされます。

4D HTMLタグとURLで利用可能

特別な**4DACTION** URLや**4DSCRIPT**、**4DTEXT**そして**4DHTML**タグを使用すると、Webに公開された4Dデータベースのプロジェクトメソッドを実行できます。例えばリクエストhttp://www.server.com/4DACTION/Erase_All は Erase_All プロジェクトメソッドが存在すればそれを実行します。

このメカニズムは特にインターネット上のユーザーが故意に (あるいは予期せず) 、Web用でないメソッドを実行してしまうという、データベースのセキュリティ上のリスクがあります。このリスクは以下の3つの方法で回避できます:

- 4D パスワードシステムを使用して、プロジェクトメソッドへのアクセスを制限する。欠点: この方法は4Dパスワードシステムを使用する必要があり、すべてのタイプのメソッド実行を禁止します (HTMLタグの利用を含む)。
- On Web Authenticationデータベースメソッドを使用してURLから呼び出されるメソッドをフィルターする。欠点: データベースに数多くのメソッドが定義されている場合、この方法は管理が困難になります。
- **4D HTMLタグとURL (4DACTION...) で利用可能**オプション (メソッドプロパティ) を使用する:


Method Properties
Access and Owner
Attributes
Cancel
OK

Name: Erase_All
Access: <Everybody>
Owner: <Everybody>
☐ Invisible
☐ Available through 4D HTML tags and URLs (4DACTION...)
☐ Offered as a Web Service
☐ Published in WSDL
☐ Shared by components and host database
☐ Available through SQL
☐ Execute on Server

このオプションを使用して、特別なURL 4DACTIONや4DSCRIPT、4DTEXT、4DHTMLタグを使用して、呼び出されるプロジェクトメソッドごとに、それを許可するかしないか設定できます。このオプションがチェックされていないと、そのプロジェクトメソッドは特別なURLやタグなどを使用したHTTPリクエストからは呼び出せません。他方、これらのメソッドを他のタイプの呼び出しでは使用することができます (フォーミュラや他のメソッドからの呼び出しなど)。

このオプションはデフォルトでチェックされていません。4DACTION Web URLや4DSCRIPT、4DTEXT、4DHTMLタグなどを使用して呼び出すメソッドは、明示的に指定する必要があります。

このプロパティが指定されたプロジェクトメソッドは、エクスプローラーで以下のアイコンが表示されます:

"4DSYNC" URLを許可する

データベース設定の"Web/設定"ページにあるこのオプションを使用して、/4DSYNC URLを含むリクエストのサポートを制限できます。このURLはHTTPを使用したデータの同期に使用されます (このメカニズムに関する詳細は4DSYNC/ URLを参照してください)。

このオプションを使用すれば/4DSYNC URLを含むリクエストの処理を有効/無効にできます:

- 選択されていない場合、/4DSYNC リクエストは標準のリクエストとみなされ、特別な処理を行いません。同期リクエストを行おうとすると"404 - resource unavailable"タイプのレスポンスが返されます。
- 選択されている場合、同期メカニズムが有効となります。/4DSYNC リクエストは特別なリクエストとして扱われ、4D HTTPサーバーが自動で解析とレスポンスを処理します。

デフォルトで:

- バージョン13以降で作成されたデータベースでは、このオプションは選択されていません。
- 互換性のため、以前のバージョンから変換されたデータベースではこのオプションが選択されています。HTTP同期機能を使用しない場合、このオプションの選択を解除してください。

このオプションのスコープはローカルアプリケーションであり、設定を変更した後はWebサーバーを再起動しなければなりません。

On Web Authenticationデータベースメソッド

説明

On Web AuthenticationデータベースメソッドはWebサーバーエンジンへのアクセス管理を担当します。このデータベースメソッドは、Webブラウザーからのリクエストがサーバー上の4Dメソッド (**4DACTION** URLや**4DSCRIPT** などを使用して呼び出されるメソッド) の実行を必要とするとき、4Dから呼ばれます。

このメソッドは6つのテキスト引数$1, $2, $3, $4, $5, $6を受け取り、ブール値を$0に返します。これらの引数の意味は以下のとおりです:

引数	型	説明
$1	テキスト	URL
$2	テキスト	HTTPヘッダー + HTTPボディ (32 KBまで)
$3	テキスト	Webクライアント (ブラウザー) のIPアドレス
$4	テキスト	サーバーのIPアドレス
$5	テキスト	ユーザー名
$6	テキスト	パスワード
$0	ブール	True = リクエストを受け入れる, False = リクエストを拒否する

これらの引数を以下のように宣言しなければなりません:

```
  ` On Web Authentication データベースメソッド
 C_TEXT($1;$2;$3;$4;$5;$6)
 C_BOOLEAN($0)
  ` メソッドコード
```

注: On Web Authenticationデータベースメソッドのすべての引数が必ず値を受け取るわけではありません。データベースメソッドが受け取る情報はデータベース設定ダイアログボックスでの設定により異なります (接続セキュリティ参照)。

- **URL**

  最初の引数 ($1) はユーザーがWebブラウザーのアドレスエリアに入力した**URL** (からホストのアドレスを取り除いたもの) です。

  イントラネット接続の場合をみてみましょう。4D WebサーバーのIPアドレスを123.4.567.89とします。以下の表はWebブラウザーに入力されたURLにより、$1が受け取る値を示しています:

Webブラウザーのアドレスに入力された値	$1の値
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers/Add	/Customers/Add
123.4.567.89/Do_This/If_OK/Do_That	/Do_This/If_OK/Do_That

- **HTTPリクエストのヘッダーとボディ**

  2番目の引数 ($2) はWebブラウザーから送信されたHTTPリクエストのヘッダーとボディです。この情報は**On Web Authenticationデータベースメソッド**にそのまま渡されることに留意してください。その内容は接続を試みたWebブラウザーの仕様により異なります。

  アプリケーションでこの情報を使用するには、開発者がヘッダーとボディを解析しなければなりません。

  注:
  - パフォーマンスの理由から、このデータのサイズは32KBで切り取られます。
  - この引数に関する詳細はOn Web Connectionデータベースメソッドの説明を参照してください。

- **WebクライアントのIPアドレス**

  $3 引数はブラウザマシンのIPアドレスを受け取ります。この情報を使用して、イントラネットアクセスとインターネットアクセスを区別できます。

- **サーバーIPアドレス**

  $4 引数はWebサーバを呼び出すために使用されたIPアドレスを受け取ります。4Dバージョン6.5以降マルチホーミングがサポートされ、複数のIPアドレスを持つマシンで使用できるようになりました。詳細はQR DELETE COLUMNを参照してください。

- **ユーザー名とパスワード**

  $5 と $6 引数は、ブラウザーが表示する標準の認証ダイアログにユーザーが入力したユーザー名とパスワードを受け取ります。このダイアログはデータベース設定でパスワード管理オプションが指定されていると、接続のたびに表示されます (接続セキュリティ参照)。

注: ブラウザーから送信されたユーザー名が4Dに存在する場合、$6 引数 (ユーザーパスワード) はセキュリティのため渡されません。

- **$0 引数On Web Authenticationデータベースメソッド**はブール値を$0に返します:
  - $0=**True**: 接続を受け入れる
  - $0=**False**: 接続を受け入れない

On Web Connectionデータベースメソッドは、接続が**On Web Authenticationデータベースメソッド**により受け入れられた時にのみ実行されます。

警告: $0に値が設定されないか、$0が**On Web Authenticationデータベースメソッド**内で定義されていない場合、接続は受け入れるものとされ、On Web Connectionデータベースメソッドが実行されます。

注:

- **On Web Authenticationデータベースメソッド**内でインターフェース要素を呼び出さないでください (ALERT, DIALOG等)。そうでなければメソッドの実行が中断され、接続は拒否されます。処理中にエラーが発生した場合も同じことが言えます。
- メソッドプロパティダイアログオックスのオプション"4DタグとURL (4DACTION...) で利用可"を使用して、プロジェクトメソッドごとに、**4DACTION** や **4DSCRIPT**にからメソッドを実行させないようにできます。この点に関する詳細は接続セキュリティを参照してください。

On Web Authentication データベースメソッドの呼び出し

On Web Authenticationデータベースメソッドは、リクエストや処理が4Dメソッドの実行を必要とするとき自動で呼び出されます。またWebサーバーが無効なスタティックURLを受け取ったときにも呼び出されます (例えばリクエストされたスタティックページが存在しない場合)。

まとめると**On Web Authenticationデータベースメソッド**は以下のケースで呼び出されます:

- 4Dが /4DACTION/ で始まるURLを受信したとき。
- 4Dが /4DCGI/ で始まるURLを受信したとき。
- 4Dが /4DSYNC/ で始まるURLを受信したとき。
- 4Dが、存在しないスタティックページをリクエストするURLを受信したとき。
- 4Dがセミダイナミックページで 4DSCRIPT タグを処理するとき。
- 4D がセミダイナミックページでメソッドに基づく4DLOOP タグを処理するとき。

互換性に関する注意: このデータベースメソッドは4Dが/4DMETHOD/ で始まるURLを受信したときにも実行されます。このURLは廃止予定であり、互換性の目的で保持されています。

有効なスタティックページをリクエストするURLを受信したとき、**On Web Authenticationデータベースメソッド**は呼び出されないことに注意してください。

例題 1

BASIC認証モードの**On Web Authenticationデータベースメソッド**の例題:

```
  `On Web Authentication データベースメソッド
 C_TEXT($1;$2;$3;$4;$5;$6)
 C_BOOLEAN($0)

 C_TEXT($user;$password;$BrowserIP;$ServerIP)
 C_BOOLEAN($4Duser)
 ARRAY TEXT($users;0)
 ARRAY LONGINT($nums;0)
 C_LONGINT($upos)

 $0:=False

 $user:=$5
 $password:=$6
 $BrowserIP:=$3
 $ServerIP:=$4

  `セキュリティのため@を含むユーザー名とパスワードは拒否
 If(WithWildcard($user)|WithWildcard($password))
    $0:=False
  `WithWildcard メソッドは後述
 Else
  `4Dユーザーかチェック
    GET USER LIST($users;$nums)
    $upos:=Find in array($users;$user)
    If($upos >0)
       $4Duser:=Not(Is user deleted($nums{$upos}))
    Else
       $4Duser:=False
    End if

    If(Not($4Duser))
  `4Dに定義されたユーザーでない場合、Webusersテーブルを検索
       QUERY([WebUsers];[WebUsers]User=$user;*)
       QUERY([WebUsers]; & [WebUsers]Password=$password)
       $0:=(Records in selection([WebUsers])=1)
    Else
       $0:=True
    End if
 End if
  `イントラネット接続か?
 If(Substring($BrowserIP;1;7)#"192.100.")
    $0:=False
 End if
```

例題 2

DIGEST認証モードの**On Web Authenticationデータベースメソッド**の例題:

```
  // On Web Authentication Database Method
 C_TEXT($1;$2;$5;$6;$3;$4)
 C_BOOLEAN($0)

 C_TEXT($user)

 $0:=False
 $user:=$5

  // セキュリティのため@文字を含むユーザー名を拒否
 If(WithWildcard($user))
    $0:=False
  // <span class="rte4d_met">WithWildcard</span> メソッドは後述
 Else
    QUERY([WebUsers];[WebUsers]User=$user)
    If(OK=1)
       $0:=WEB Validate digest($user;[WebUsers]password)
    Else
       $0:=False // ユーザーが存在しない
    End if
 End if
```

WithWildcard メソッドは以下の通りです:

```
  // WithWildcard メソッド
  // WithWildcard ( String ) -> ブール
  // WithWildcard ( Name ) -> ワイルドカード文字を含んでいるか

 C_INTEGER($i)
 C_BOOLEAN($0)
 C_TEXT($1)

 $0:=False
 For($i;1;Length($1))
    If(Character code(Substring($1;$i;1))=Character code("@"))
       $0:=True
    End if
 End for
```

On Web Connectionデータベースメソッド

On Web Connectionデータベースメソッドは以下のケースで呼び出されます:

- Webサーバが /4DCGI/ から始まるURLを受信した。
- Webサーバが無効なリクエストを受信した。

詳細な情報は、後述の"On Web Connection データベースメソッド呼び出し" の段落を参照してください。

互換性に関する注意: コンテキストモードでコンテキストが作成されたときもデータベースメソッドは呼び出されます。コンテキストモードは廃止予定のモードで、変換されたデータベースで利用できます。

データベースがWebサーバとして公開され、リクエストは事前にOn Web Authenticationデータベースメソッドで受け入れられていなければなりません(存在する場合)。

On Web Connectionデータベースメソッドは6つのテキスト引数を受け取ります。これらの引数の内容は以下のとおりです:

引数	型	説明
$1	テキスト	URL
$2	テキスト	HTTPヘッダ + HTTPボディ (32 KBまで)
$3	テキスト	Webクライアント (ブラウザ) のIPアドレス
$4	テキスト	サーバのIPアドレス
$5	テキスト	ユーザ名
$6	テキスト	パスワード

これらの引数を以下のように宣言しなければなりません:

```
  // On Web Connection データベースメソッド
 C_TEXT($1;$2;$3;$4;$5;$6)
  // メソッドコード
```

- **URL**

  最初の引数 ($1) は、ユーザがWebブラウザのアドレスエリアに入力した**URL**からホストのアドレスを取り除いたものです。
  イントラネット接続の場合をみてみましょう。4D WebサーバのIPアドレスを123.4.567.89とします。以下の表はWebブラウザに入力された**URL**に対して、$1が受け取る値を示しています:

Webブラウザのアドレスに入力された値	$1の値
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers/Add	/Customers/Add
123.4.567.89/Do_This/If_OK/Do_That	/Do_This/If_OK/Do_That

  この引数は必要に応じて自由に利用できます。4Dは単にURLのホスト部より後の部分を$1に渡します。
  例えば値 "/Customers/Add" が "直接新規レコードを[Customers] テーブルに追加する"ということを意味するというような、オリジナルのルールを作成できます。Webユーザにデータベースを公開し、利用可能な値やブックマークを提供できます。アプリケーションの異なる部分へのショートカットを提供できます。このようにして、Webユーザはデータベースに接続するたびにナビゲーションを通過することなく、素早くWebサイトのリソースにアクセスできます。

警告: 以前のセッションで作成されたブックマークでデータベースに再入力されることを防ぐため、4Dは標準の4D URLに対応するURLをすべてキャッチします。

- **HTTPリクエストのヘッダとボディ**

  2番目の引数 ($2) はWebブラウザから送信されたHTTPリクエストのヘッダとボディです。この情報は**On Web Connectionデータベースメソッド**にそのまま渡されることに留意してください。その内容は接続を試みたWebブラウザの仕様により異なります。
  Mac OS上のSafariでは、以下のようなヘッダを受け取るでしょう (一部省略):

  ```
  GET /favicon.ico HTTP/1.1
  Referer: http://123.45.67.89/4dcgi/test
  User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X; fr-fr) AppleWebKit/523.10.3 (KHTML, like Gecko)
  Version/3.0.4 Safari/523.10
  Cache-Control: max-age=0
  Accept: */*
  Accept-Language: ja-jp
  Accept-Encoding: gzip, deflate
  Connection: keep-alive
  Host: 123.45.67.89
  ```

  Windows上のMicrosoft Internet Explorer 8では、以下のようなヘッダを受け取るでしょう:

  ```
  GET / HTTP/1.1
  Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml, image/pjpeg,
  application/x-ms-xbap, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
  Accept-Language: ja-JP
  User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET
  CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C)
  Accept-Encoding: gzip, deflate
  Host: 123.45.67.89
  Connection: Keep-Alive
  ```

  アプリケーションでこの情報を使用するには、開発者がヘッダとボディを解析しなければなりません。
  注: パフォーマンスの理由から、このデータのサイズは32KBで切り取られます。

- **WebクライアントのIPアドレス**

  $3 引数はブラウザマシンのIPアドレスを受け取ります。この情報を使用して、イントラネットアクセスとインターネットアクセスを区別できます。

- **サーバIPアドレス**

  $4 引数はWebサーバを呼び出すために使用されたIPアドレスを受け取ります。4Dバージョン6.5以降マルチホーミングがサポートされ、複数のIPアドレスを持つマシンで使用できるようになりました。詳細はWebサーバー設定を参照してください。

- **ユーザ名とパスワード**

  $5 と $6 引数は、ブラウザが表示する標準の認証ダイアログにユーザが入力したユーザ名とパスワードを受け取ります。このダイアログは環境設定でパスワード管理オプションが指定されていると、接続のたびに表示されます (接続セキュリティ参照)。

注: ブラウザから送信されたユーザ名が4Dに存在する場合、$6 引数 (ユーザパスワード) はセキュリティのため渡されません。

On Web Connection データベースメソッドの呼び出し

は4DCGI URLまたはカスタマイズされたコマンドURLを使用したWebサーバーへのアクセスのエントリポイントとして使用できます。

警告: インタフェース要素を表示する4D コマンド (ALERT, DIALOG...) を呼び出すと、メソッド処理が終了します。

は以下のケースで呼び出されます:

- 4Dが */4DCGI/* URLを受け取ったとき。$1に */4DCGI/<action>* が渡されて、データベースメソッドが呼び出されます。
- *<path>/<file>*タイプのURLで存在しないWebページが呼び出されたとき。データベースメソッドにそのURLが渡されて呼び出されます (*)。
- *<file>/*タイプのURLでWebページが呼び出され、デフォルトのホームページが設定されていないとき。データベースメソッドにそのURLが渡されて呼び出されます (*)。

(*) 特定のケースでは、$1に渡されるURLはスラッシュ"/"で始まりません。

On Web Session Suspend データベースメソッド

On Web Session Suspend データベースメソッド はWebセッションが閉じられる直前に、4D Webサーバーから呼び出されます。4Dは以下のような場合にWebセッション (セッションを管理するWebプロセス) を閉じます:

- セッションを管理するWebプロセス数の最大値 (デフォルトで100、WEB SET OPTIONコマンドで変更可能) に達している状態で、さらに新しいWebセッションを作成する必要があるとき (4Dは一番古いWebセッションプロセスを自動で破棄します)
- セッションプロセスのタイムアウトに達したとき (デフォルトで480分 = 8時間、WEB SET OPTIONコマンドで変更可能)
- WEB CLOSE SESSIONコマンドが呼び出された場合

このデータベースメソッドが呼び出された時点で、セッションのコンテキスト (プロセス変数の値やカレントセレクション) は有効です。そのセッションに関連するデータ (変数の値やセレクション) を退避し、後で同じcookie値でリクエストを受信したときにそれらを再利用することができます。

On Web Session Suspend データベースメソッドの例題はWebセッション管理を参照してください。

Webセッション管理

4D Webサーバーはユーザーセッションを容易に管理するメカニズムを提供します。この自動メカニズムを使用して、後のリクエストでWebクライアントが同じコンテキスト (セレクションや変数の値) を再利用できます。

このメカニズムでは4Dが自動で設定するプライベートな"4DSID"という名前のcookieを使用します (cookie名は変更できます)。Webクライアントからリクエストを受信するたびに4Dは4DSID cookieが送信されてきたか、またその値をチェックします:

- cookieの値がリクエストヘッダーに含まれていれば、4Dは存在するWebプロセスの中から対応するプロセスを検索します。
  - プロセスが見つかれば、そのプロセスがWebリクエストの処理に使用されます。Compiler_Web メソッドは実行されません。
  - プロセスが見つからない場合、4Dは新しいプロセスを作成します。
- cookieがリクエストヘッダーに含まれていなければ、4Dは新しいコンテキストを作成します。

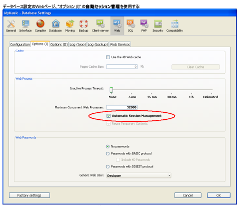
自動セッション管理を有効/無効にする

アプリケーションで4Dの自動Webセッション管理を使用するには、その機能が有効化されていなければなりません。

4D v13以降で作成されたデータベースでは、このメカニズムがデフォルトで有効になっています。4D v12以前から変換されたデータベースの場合、互換性のためこの機能は無効になっています。この新機能を使用するためには明示的に有効化しなければなりません。

自動セッション管理を有効化する方法は2つあります:

- データベース設定のWebページ、"オプション (I)" の**自動セッション管理**を使用する:



☐ Use the 4D Web cache

Pages Cache Size: 0 Kb

Clear Cache

Inactive Process Timeout: None

Maximum Concurrent Web Processes: 32000

☑ Automatic Session Management

☑ Reuse Temporary Contexts

● No passwords

○ Passwords with BASIC protocol

☐ Include 4D Passwords

○ Passwords with DIGEST protocol

Generic Web User: Designer

  これを使用した場合、設定はストラクチャーに保存され永続化されます。
- WEB SET OPTIONコマンドでWeb Keep sessionオプションを使用する。この場合、設定は4Dを終了するまで有効です。

いずれの場合も設定はローカルマシンに有効です。つまり4D Serverとリモートモードの4Dでは異なる設定を行うことができます。

cookie の有効期限とコンテキストの保存

cookieの有効期限およびセッションに割り当てられたWebプロセスのタイムアウトはデフォルトで480分 (8時間) です。これらの有効期限やタイムアウトはWEB SET OPTIONコマンドを使用して個別に変更できます。cookieの有効期限は Web Inactive session timeout オプション、Webプロセスのタイムアウトは Web Inactive process timeout オプションで設定します。例えばブラウザー側でショッピングカート用のcookie有効期限を24時間有効にするとしても、最適化の目的でWebプロセス をそんなに長く保持したくないと思うでしょう。この場合、Webプロセスのタイムアウトをもっと短く設定することができます。Webプロセスがタイムアウ トするときはOn Web Session Suspend データベースメソッドが呼び出され、プロセスが破棄される前にセレクションや変数の値などを後で利用するために退避させることができます。次回Webク ライアントが (24時間以内に) 接続してきたときは、同じcookieがサーバーに送信されるので、そのcookie値をキーとして前回のセッションのコンテキスト (変数値やセレクションの状態など) を On Web Connectionデータベースメソッド 内に読み込むことができます (後述の例題参照)。

必要であれば、WEB CLOSE SESSIONコマンドを使用していつでもcookieを強制的に失効させ、セッションを閉じることができます。

非動作コンテキストの破棄

4Dはセッションを管理するWebプロセス数が上限値に達した場合、もっとも古いプロセスを破棄します。この上限値はデフォルトで100であり、WEB SET OPTIONコマンドの Web Max sessions オプションで変更できます。
プロセスが破棄される直前にはOn Web Session Suspend データベースメソッドが呼び出されるので、変数やセレクションなどを次回のリクエストで再利用できるよう退避させることができます。

例題

この例題ではOn Web Connectionデータベースメソッドや On Web Session Suspend データベースメソッドを使用してセッションを管理する方法を示します。

以下はOn Web Connectionデータベースメソッドのコードです:

```
  // On Web Connection (または On Web Authentication)
 C_TEXT(www_SessionID)
 If(www_SessionID=WEB Get Current Session ID)
  // Compiler_Webは呼び出されない
  // すべての変数やセレクションは既に存在している
    ...
 Else
  // Compiler_Web が実行された
  // これは新しいセッションで、変数やセレクションは存在しない
  // 4Dが作成したこのセッションのIDをプロセス変数に格納する
    www_SessionID:=WEB Get Current Session ID

  // Webプロセスの初期化を行う
  // セレクションを作成
  // ユーザーを検索
    QUERY([user];[user]Login=www_Login)
    QUERY([prefs];[prefs]Login=www_Login)

  // 従業員情報を検索
    QUERY([employees];[employees]Name=[user]name)
    QUERY([company];[company]Name=[user]company)

  // 変数のセットアップ
  // このユーザーの環境設定をロード
    SELECTION TO ARRAY([prefs]name;prefNames;[prefs]value;prefValues)
    www_UserName:=[User]Name
    www_UserMail:=[User]mail

  // セッションが初期化された
 End if
```

以下はOn Web Session Suspend データベースメソッドのコードです :

```
  // On Web Session Suspend
  // タイムアウトその他の理由で4Dはセッションを閉じることがある
 C_TEXT(www_SessionID)
 www_SessionID:=""
  // セッション情報を永続化する
  // このプロセスを使用して接続していたユーザーの環境設定を保存
 QUERY([prefs];[prefs]Login=www_Login) // 変数値はまだ使用できる
 ARRAY TO SELECTION(prefNames;[prefs]name;prefValues;[prefs]value)

  // 重要: 4Dはプロセスを終了します
  // 変数やセレクションなどは破棄されます。
```

COOKIEが拒否されている場合

4Dの自動セッション管理メカニズムはcookieを利用しています。Webクライアントがcookieをサポートしない、あるいは拒否している場合、4D HTTPサーバーはセッションを保守できません。この場合各リクエストは新規接続として扱われ、各接続毎に Compiler_Web メソッドが実行されます。

セッションとIPアドレス

4D HTTPサーバーはセッションを開始したIPアドレスを記憶します。異なるIPアドレスから既存のセッションにアクセスされた場合、HTTP 400エラーがクライアントに返されます。

4DオブジェクトをHTMLオブジェクトにバインドする

この節では4D Webサーバーにおいて、ダイナミックな値を送受信するなどの情報交換を行うための方法を説明します。以下のポイントが関連します:

- 4D変数に格納された値を送信する
- Webフォームから送信された値を受信する
- JavaScriptの有効化

ダイナミックな値の送信

4D変数への参照をHTMLページに挿入できます。この参照をHTMLオブジェクトにバインドできます。Webページがブラウザーに送信されるとき、4Dはこれらの参照を変数の現在の値で置き換えます。ブラウザが受け取るページは、スタティックな要素と、4D由来の値の組み合わせとなります。このタイプのページをセミダイナミックと呼びます。

注:

- プロセス変数が使用できます。
- HTMLはワープロ志向の言語なので、通常テキスト変数を使用します。しかしBLOB変数を使用することもできます。そのためにはテキストを長さ情報なしのBLOBにします。

まず、HTMLオブジェクトは4D変数の値を使用して初期化された値を持ちます。

2番目に、Webフォームがサーバーに送信されると、HTMLオブジェクトの値が4D変数に戻されます。これを行うには、フォームのHTMLソースで、バインドさせたい**4Dプロセス変数と同じ名前を持つ**HTMLオブジェクトを作成します。この点は後ほど"ダイナミックな値の受信"で説明します。

注: 4Dピクチャー変数への参照を作成することはできません。

HTMLオブジェクトの値を4D変数で初期化できるので、HTMLオブジェクトの**value** フィールドに **<!--#4DTEXT VarName-->** を記述することによって、プログラムでデフォルト値を提供できます。VarName は、カレントのWebプロセスで定義された4Dプロセス変数名です。この名前はHTML標準のコメント記法に囲まれています。

注: HTMLエディタによっては **<!--#4DTEXT VarName-->** をHTMLオブジェクトの **value** フィールドに許可しないかもしれません。この場合、HTMLコードに直接入力する必要があります。

<!--#4DTEXT --> タグを使用すればページに**4D式** (フィールドや配列要素) を挿入することもできます。 このタイプのデータに対するこのタグの動作は、変数のそれと同じです。詳細はST FREEZE EXPRESSIONSを参照してください。

実際、シンタックス **<!--#4DTEXT VarName-->** は、HTMLページのどこにでも、4Dデータを挿入することを可能にします。例えば以下のように記述すると:

<P>Welcome to <!--#4DTEXT vtSiteName-->!</P>

4D変数vtSiteNameの値がHTMLページに挿入されます。

例題:

```
  ` The following piece of 4D code assigns "4D4D" to the process variable vs4D
 vs4D:="4D4D"
  ` Then it send the HTML page "AnyPage.HTM"
 SEND HTML FILE("AnyPage.HTM")
```

AnyPage.HTMのソースは以下のようなものです:

```
<html>
<head>
    <title>AnyPage</title>
    <script language="JavaScript">
        function Is4DWebServer(){
            return (document.frm.vs4D.value=="4D4D")
        }

        function HandleButton(){
            if(Is4DWebServer()){
                alert("You are connected to 4D Web Server!")
            } else {
                alert("You are NOT connected to 4D Web Server!")
            }
        }
    </script>
</head>
<body>
    <form action="/4DACTION/WWW_STD_FORM_POST" method="post" name="frm">
        <p><input type="hidden" name="vs4D" value="<!--#4DTEXT vs4D-->"</p>
        <p><a href="JavaScript:HandleButton()"><img src="AnyGIF.GIF" border=0 align=bottom></a></p>
        <p><input type="submit" name="bOK" value="OK"></p>
    </form>
</body>
</html>
```

サーバから送信されるページの解析

最適化の目的で、URLで呼び出されたHTMLページの拡張子が"HTML"や"HTM"の場合、4D WebサーバーはHTMLソースコードの解析を行いません。もちろん必要に応じてページの解析を"強制的に"行わせるメカニズムも提供されています (4D HTMLタグ参照)。

4D変数にHTMLコードを挿入

4D変数にHTMLコードを挿入できます。HTMLスタティックページがWebブラウザに表示されるとき、変数の値はHTMLコードに置き換えられ、ブラウザに解釈されます。

4D変数や式を使用してHTMLコードを挿入するために、特別な4DHTMLタグを使用できます:

例えば以下の4D変数があるとき:

```
 vtHTML:="<b>"+[Client]name+"</b>"
```

以下のコメントを使用してHTMLページにHTMLコードを挿入できます:

```
<!--#4DHTML vtHTML-->
```

テキストや (Text without lengthモードで生成された) BLOBタイプの変数を使用できます。

詳細はST FREEZE EXPRESSIONSを参照してください。

ダイナミックな値の受信

4D server がポストされたフォームを受信すると、4Dはそれに含まれるどんなHTMLオブジェクトの値も受け取ることができます。この原理はユーザーが値を入力または修正して、評価ボタンをクリックするという WEB SEND FILE コマンドまたは WEB SEND BLOB コマンドの例題で送信されたWebフォームにおいても使用可能です。この場合、4DがリクエストにHTMLオブジェクトの値を取得する方法は二つあります:

- WEB GET VARIABLES コマンドを使用
- または、WEB GET BODY PART コマンドと WEB Get body part countコマンドを使用

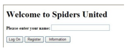
WEB GET VARIABLES コマンドは値をテキストとして受け取るのに対し、WEB GET BODY PART コマンドと WEB Get body part count コマンドはBLOBを使用してポストされたファイルを取得します。

互換性の注意 (4D v13.4): 以前のバージョンでは、4DはHTTP フォームまたはGET type URLを使用してポストされた変数の値を、4Dプロセス変数へとコピーしていました。(コンパイルモードでは、変数は *COMPILER_WEB* プロジェクトメソッドにて事前に宣言されていなければなりませんでした)。この機能は4D v13.4以降削除されており、変換されたデータベースにおいては互換性のために維持はされているものの、データベース設定の 互換性ページ 内の**Web変数に値を自動的に代入する**のオプションのチェックを外すことによって無効化することができます(このチェックは外して WEB GET VARIABLES コマンドまたは WEB GET BODY PART コマンドを使用することが推奨されます)。

以下のHTMLソースコードがあるとき:

```
<html>
<head>
  <title>Welcome</title>
  <script language="JavaScript"><!--
function GetBrowserInformation(formObj){
formObj.vtNav_appName.value = navigator.appName
formObj.vtNav_appVersion.value = navigator.appVersion
formObj.vtNav_appCodeName.value = navigator.appCodeName
formObj.vtNav_userAgent.value = navigator.userAgent
return true
}
function LogOn(formObj){
if(formObj.vtUserName.value!=""){
return true
} else {
alert("Enter your name, then try again.")
return false
}
}
//--></script>
</head>
<body>
<form action="/4DACTION/WWW_STD_FORM_POST" method="post"
 name="frmWelcome"
 onsubmit="return GetBrowserInformation(frmWelcome)">
  <h1>Welcome to Spiders United</h1>
  <p><b>Please enter your name:</b>
  <input name="vtUserName" value="" size="30" type="text"></p>
  <p>
<input name="vsbLogOn" value="Log On" onclick="return LogOn(frmWelcome)" type="submit">
<input name="vsbRegister" value="Register" type="submit">
<input name="vsbInformation" value="Information" type="submit"></p>
<p>
<input name="vtNav_appName" value="" type="hidden">
<input name="vtNav_appVersion" value="" type="hidden">
<input name="vtNav_appCodeName" value="" type="hidden">
<input name="vtNav_userAgent" value="" type="hidden"></p>
</form>
</body>
</html>
```

4DがWebブラウザにページを送信すると、以下のように表示されます:



Welcome to Spiders United

Please enter your name:

このページの主な特徴は:

- 送信のためのボタンが3つ(vsbLogOn, vsbRegister そして vsbInformation)があるということ
- Log Onをクリックすると、フォームからの送信はまず初めに JavaScript ファンクションの LogOn によって処理されます。名前が何も入力されていない場合、フォームは4Dに送信すらされず、JavaScript警告が表示されます。
- フォームにはPOST 4D メソッドに加えて、Navigator のプロパティをvtNav_App から始まる名前の4つの隠しオブジェクトへとコピーする投稿スクリプト (GetBrowserInformation) を持っています。
- また、このページにはvtUserName オブジェクトも含まれます。

ユーザーがHTMLフォーム上のボタンのどれかをクリックした際に呼び出される WWW_STD_FORM_POST という 4D メソッドについて検証します。

```
  // 変数の値の取得
 ARRAY TEXT($arrNames;0)
 ARRAY TEXT($arrValues;0)
 WEB GET VARIABLES($arrNames;$arrValues)
 C_TEXT($user)

 Case of

  // Log On ボタンがクリックされる
    :(Find in array($arrNames;"vsbLogOn")#-1)
       $user :=Find in array($arrNames;"vtUserName")
       QUERY([WWW Users];[WWW Users]UserName=$arrValues{$user})
       $0:=(Records in selection([WWW Users])>0)
       If($0)
          WWW POST EVENT("Log On";WWW Log information)
  // WWW POST EVENT メソッドによって情報がデータベースのテーブルへと保存される
       Else

          $0:=WWW Register
  // WWW Register メソッドによって新しいWebユーザーが登録可能
       End if

  // Register ボタンがクリックされる
    :(Find in array($arrNames;"vsbRegister")#-1)
       $0:=WWW Register

  // Information ボタンがクリックされる
    :(Find in array($arrNames;"vsbInformation")#-1)
       WEB SEND FILE("userinfos.html")
 End case
```

このメソッドの特徴は:

- 変数vtNav_appName, vtNav_appVersion, vtNav_appCodeName, そして vtNav_userAgent の値(HTMLオブジェクトと同じ名前を持つことでそれぞれバインドされています)は、WEB GET VARIABLES コマンドを使用することによって GetBrowserInformation JavaScript のスクリプトで作成された HTML オブジェクトから取得することができます。

- 3つの投稿ボタンにバインドされている変数vsbLogOn, vsbRegister と vsbInformation のうち、クリックされたボタンに対応するもののみが WEB GET VARIABLES コマンドによって取得されます。この3つのうちどれかのボタンによって投稿が行われたとき、ブラウザはクリックされたボタンの値を4Dに返します。これによりどのボタンがクリックされたのが分かります。4Dフォーム側の4Dボタンは数字変数であることに留意して下さい。しかしながら、HTMLでは全てのオブジェクトはテキストオブジェクトです。

SELECT objectを使用した場合、WEB GET VARIABLES コマンドで返されるのはオブジェクト内でハイライトされている要素の値であり、4Dのように要素の配列内の位置を返すわけではないという事に留意して下さい。

WEB GET VARIABLES コマンドは必ずテキスト型の値を返します。

COMPILER_WEB プロジェクトメソッド

ポストされたフォームを4D Webサーバーが受け取る際、COMPILER_WEBプロジェクトメソッドがあれば、自動で呼び出されます。このメソッドには変数の型宣言と変数の初期化コードを記述します。変数名はHTMLフォームのフィールド名と同じものです。データベースをコンパイルする際には、このメソッドが使用されます。

COMPILER_WEB はすべてのWebフォームで共通です。デフォルトでCOMPILER_WEBメソッドは存在しません。明示的に作成する必要があります。

注: WEB GET VARIABLESコマンドを使用して、サブミットされたHTMLページに含まれる値を取得することもできます。

Web サービス: COMPILER_WEB は、存在すれば、SOAPリクエストが受け入れられるごとに実行されます。このメソッドを使用して、Webサービスとして公開されたすべてのメソッドで、受信するSOAP引数に割り当てられたすべての4D引数を宣言しなければなりません。実際、Webサービスメソッドで引数の受け取りにプロセス変数を使用 する場合、メソッドが呼び出される前にそれらが宣言されていなければなりません。この点に関する詳細はSOAP DECLARATIONコマンドの説明を参照してください。

JavaScript のカプセル化

4DはJavaScriptソースコードとJavaScript .js ファイル (例 <SCRIPT SRC="...")とをHTML ドキュメントに埋め込むことをサポートしています。

WEB SEND FILE や WEB SEND BLOBを使用すると、HTMLソースエディタで編集したHTMLソー スや4Dからディスクに保存したHTMLを送信します。両方のケースで、ページをフルコントロールできます。ドキュメントのHEAD部に JavaScriptスクリプトを挿入したり、FORMマークアップでスクリプトを使用したりできます。先の例題で、フォームに名前を付け、スクリプトか らそのフォームを参照しました。またフォームのマークアップレベルでサブミットを受け入れたり拒否したりできます。

注: 4DはJava applets transportをサポートしています。

URLとフォームアクション

4D Webサーバーはデータベースに様々なアクションを実装することを可能にする、URLやHTMLフォームアクションを提供します。

URLには以下のものがあります:

- **/4DACTION/** はHTMLオブジェクトとデータベースのプロジェクトメソッドをリンクします。
- **/4DCGI/** はHTMLオブジェクトからOn Web Connectionデータベースメソッドを呼び出します。
- **/4DSYNC/** はテーブルのデータを同期するために使用します。

さらに4D Webサーバーは追加でいくつかのURLを受け入れます:

- **/4DSTATS**, **/4DHTMLSTATS**, **/4DCACHECLEAR** そして **/4DWEBTEST**は、4D Webサイトの動作状況に関する情報を入手するために使用します。これらのURLはWebサイトに関する情報で説明しています。
- **/4DWSDL**は、Webサービスサーバー上で公開されているWebサービスの定義ファイルへのアクセスを可能にします。詳細情報はWebサービス(サーバ) コマンドとDesign Referenceマニュアルを参照してください。

URL 4DACTION/

シンタックス: 4DACTION/MyMethod{/Param}

利用法: URLまたはフォームアクション

このURLを使用して、HTMLオブジェクト (テキスト、ボタン...) を4Dプロジェクトメソッドにリンクできます。リンクは **/4DACTION/MyMethod/Param** のように記述され、MyMethod はユーザーがリンクをクリックしたときに実行される4Dプロジェクトメソッド名、Param はオプションのテキスト引数で、MyMethodメソッドの $1 に渡されます (後述の"URLから呼ばれる4Dメソッドに渡されるテキスト引数"参照)。
4Dが **/4DACTION/MyMethod/Param** リクエストを受け取ると On Web Authenticationデータベースメソッド が (存在すれば) 呼ばれます。このメソッドから**True**が返されると、MyMethodメソッドが実行されます。
4DACTION/ をスタティックなWebページのURLに割り当てることもできます。URLのシンタックスは以下の形式でなければなりません:

```
<a href="/4DACTION/MyMethod/Param">Do  Something</a>
```

MyMethodプロジェクトメソッドは通常レスポンスを返すべきです (WEB SEND FILE や WEB SEND BLOBでHTMLページを送信するなど)。ブラウザーをブロックしないように、処理は可能な限り短時間で行われるようにします。

注: /4DACTION/ から呼び出されるメソッドはインターフェース要素 (DIALOG, ALERT...) を呼び出してはいけません。

警告: /4DACTION/ URLを使用して4Dメソッドを呼び出せるようにするには、メソッドプロパティで"4D HTMLタグやURL (4DACTION) で利用可能" 属性がチェックされていなければなりません。これはデフォルトで選択されていません。詳細は接続セキュリティを参照してください。

例題 1

この例題はHTMLピクチャーオブジェクトに**/4DACTION/** URLを割り当て、ページにダイナミックなピクチャーを表示する方法を説明しています。スタティックHTMLページに以下のコードを記述します:

```
<img src="/4DACTION/PICTFROMLIB/1000">
```

PICTFROMLIBメソッドは以下のとおりです:

```
 C_TEXT($1) // この引数は常に宣言する
 C_PICTURE($PictVar)
 C_BLOB($BlobVar)
 C_LONGINT($Number)
  // ピクチャー番号を$1文字列から取り出す
 $Number:=Num(Substring($1;2;99))
 GET PICTURE FROM LIBRARY($Number;$PictVar)
 PICTURE TO GIF($PictVar;$BlobVar)
 WEB SEND BLOB($BlobVar;"Pict/gif")
```

4DACTION を使用してフォームをポスト

4D Webサーバーは、ポストされたフォームを使用する際に、スタティックなページからWebサーバーにデータを送信するという新たな選択肢を提供します。POSTタイプを使用し、フォームのアクションは **/4DACTION/MethodName** で始まっていなければなりません。

注: フォームは２つのメソッドを使用してサブミットできます (4Dでは両方のタイプを使用できます):

- POSTは通常Webサーバーからデータベースにデータを追加するために使用します。
- GETは通常Webサーバーに、データベースから取得するデータをリクエストするために使用します。

この場合、Webサーバーがポストされたフォームを受信すると、COMPILER_WEBプロジェクトメソッド (存在すれば、後述参照) が呼び出され、そしてOn Web Authenticationデータベースメソッドが (存在すれば) 呼び出されます。このメソッドが**True**を返すと、MethodNameメソッドが実行されます。4Dはフォームに含まれるHTMLフィールドを解析し、値を取り出して自動で4D変数に代入します。フォーム中のフィールドと4D変数は同じ名前でなければなりません。

注: 詳細情報は4DオブジェクトをHTMLオブジェクトにバインドするを参照してください。

フォームに適用するHTMLシンタックスは以下のタイプです:

- フォームのアクションを定義するには:

  ```
  <form action="/4DACTION/MethodName" method="post">
  ```

- フォームのフィールドを定義するには:

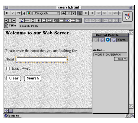
  ```
  <input type="Field_type" name="Field_name" value="Default_value">
  ```

フォーム中のフィールドごとに、4Dはフォームフィールドの値を同じ名前の変数に代入します。フォームオプション (チェックボックスなど), の場合、4Dは割り当てられた変数を、選択されていれば1に、そうでなければ0に設定します。
数値の入力の場合、4Dはフィールドの値を文字から実数に変換します。

注: フォーム中のフィールドの名前がOKであるとき (例えばサブミットボタン)、フィールド値が空でなければOKシステム変数が1に設定されます。そうでなければ0に設定されます。

例題 2

4D Webデータベースが開始されていて、ブラウザーがスタティックHTMLページからレコードを検索できるようにしたいとします。このページを"search.htm"とします。データベースには検索結果を表示するための"results.htm"のようなスタティックページもあります。 POSTメソッドと**/4DACTION/SEARCH**アクションがページに割り当てられています。



Welcome to our Web Server

Please enter the name that you are looking for.

Name:

☐ Exact Word

以下はこのページのHTMLコードです:

```
<form action="/4DACTION/PROCESSFORM" method="post">
<input type="text" name="vname" value=""><br>
<!-- 通常ボタン名をVALUEにいれますが、解釈のためVALUEに数字を入れなければいけない-->
<input type="checkbox" name="vexact" value="1">単語全体<br>
<!-- OKは特別なケース-->
<input type="submit" name="OK" value="検索">
</form>
```

データ入力エリアに"ABCD"とタイプし、オプションをチェックして検索ボタンをクリックします。

4Dは以下のCOMPILER_WEBメソッドを呼び出します:

```
 C_TEXT(VNAME)
 VNAME:=""
 C_LONGINT(vEXACT)
 vEXACT:=0
 OK:=0 `particular case
```

この例題ではvnameに文字列"ABCD"が代入され、vexactは1に、OKも1になります (ボタン名がOKであるため)。
4DはOn Web Authenticationデータベースメソッドを (存在すれば) 呼び出し、そして以下のPROCESSFORMプロジェクトメソッドが呼び出されます:

```
 If(OK=1)
    If(vEXACT=0) // オプションが選択されていなければ
       vNAME:=VNAME+"@"
    End if
    QUERY([Jockeys];[Jockeys]Name=vNAME)
    vLIST:=Char(1) // Return the list in HTML
    FIRST RECORD([Jockeys])
    While(Not(End selection([Jockeys])))
       vLIST:=vLIST+[Jockeys]Name++[Jockeys]Tel+<BR>
       NEXT RECORD([Jockeys])
    End while
    WEB SEND FILE(results.htm) // 検索結果が挿入されるresults.htmを送信
  // このページには変数vLISTの参照が含まれている (<!--4DTEXT vLIST-->)
    ...
 End if
```

URL 4DCGI/

シンタックス: 4DCGI/<action>

モード: 両方

利用法: URL.

4D Webサーバーが **/4DCGI/<action>** URLを受信すると、On Web Authenticationデータベースメソッドが (存在すれば) 呼び出されます。このメソッドが**True**を返すと、Webサーバーは On Web Connectionデータベースメソッドを呼び出し、$1にURLをそのまま渡します。

/4DCGI/ URLはどのファイルにも対応しません。その役割は4DのOn Web Connectionデータベースメソッドを呼び出すことです。"<action>" 引数にはどのようなタイプの情報でも含めることができます。

このURLを使用してどのようなタイプのアクションでも行うことができます。On Web Connectionデータベースメソッドやそのサブメソッド内で$1値 をテストして、適切なアクションを実行できます。例えば完全にカスタマイズされたHTMLからレコードの追加、検索、並び替えなどを行ったり、GIFイ メージを作成したりできます。このURLを使用する例題はPICTURE TO GIF および [#cmd id="659"/]コマンドにあります。

アクションを指示した後は、データを送信するコマンド (WEB SEND FILE, WEB SEND BLOB等) を使用してレスポンスを返さなければなりません。

警告: ブラウザーをブロックしないようにするため、アクションはなるべく短時間で終わらせるようにしてください。

URLから呼ばれる4Dメソッドに渡されるテキスト引数

4Dは特別なURL (**/4DACTION/**) から呼ばれるメソッドにテキスト引数を渡します。これらのテキスト引数に関して:

- これらの引数を使用しない場合でも、C_TEXTコマンドを使用して明示的に定義しなければなりません。そうしなければコンパイルモードで実行されているデータベースにアクセスするWebを使用すると、以下のようなランタイムエラーが発生します: **"A runtime error occurred at line number: 0 when executing the method " ON WEB CONNECTION. Invalid parameter in a Run command"**.
  このラインタイムエラーは、HTMLリンクがクリックされることで呼び出される4Dメソッド中で、$1テキスト引数が定義されていないことにより発生します。実行コンテキストはカレントのHTMLページであるため、エラーはページをWebブラウザに送信したメソッドの"line 0"を参照します。明示的に$1テキスト引数を定義することでこれらのエラーを取り除くことができます:
  [#code4D]// M_ADD_RECORDSプロジェクトメソッド
  C_TEXT($1) ` この引数を明示的に宣言しなければならない
  Repeat
  CREATE RECORD ([Clients])
  // フィールドにデータを代入
  SAVE RECORD ([Clients])
  Until (OK=0)
- $1 引数にはURLに追加されたデータが格納され、HTML環境から4D環境にデータを渡すためのプレースホルダーとして使用できます。

呼び出された4Dメソッドで明示的に宣言する必要のある引数

4Dメソッドへの呼び出し元により、異なる引数を宣言しなければなりません。

- On Web Authenticationデータベースメソッド (存在する場合)とOn Web Connectionデータベースメソッド
  接続に関する6つの引数を宣言しなければなりません:

  ```
    // On Web Connection データベースメソッド
   C_TEXT($1;$2;$3;$4;$5;$6)
  ```

- **/4DACTION/** URLから呼ばれるメソッド
  $1 引数を宣言します:

  ```
    // 4DACTION/ URLから呼ばれるメソッド
   C_TEXT($1)
  ```

- ドキュメント中のHTMLコメントとして描かれる**/4DSCRIPT/** タグから呼ばれるメソッド
  メソッドは$0にテキストを返すことができます。$0 と $1 引数を宣言しなければなりません:

  ```
    // 4DSCRIPT HTMLコメントから呼び出されるメソッド
   C_TEXT($0;$1)
  ```

URL 4DSYNC/

シンタックス:
/4DSYNC/$catalog{/TableName}
/4DSYNC/TableName{/TableName}{/FieldName1,...,FieldNameN}{Params}

利用: POSTまたはGETメソッドのURL

このURLを使用して、ローカル4Dデータベースのテーブル中のデータをリモートのデータベースとHTTPで同期できます。これは例えばスマートフォン上にインストールされたクライアントアプリケーションや、サードパーティーのHTTPアプリケーションと4Dデータベースを同期するために使用できます。

/4DSYNC/ URLをGETメソッドで使用して4Dデータベースのデータを取得したり、POSTメソッドで使用して4Dデータベースのデータを更新したりできます。

使用できるHTTPリクエストは以下の通りです:

- *GET /4DSYNC/$catalog*
  データベーステーブルのリストと、含まれるレコード数が返されます。例えば、PEOPLE（2レコード）とINVOICES（3レコード）という2つのテーブルがあるストラクチャにおいて http://localhost/4DSYNC/$catalog/ を使用したとき、PEOPLE 2 INVOICES 3がWebクライアントに返されます。

- *GET /4DSYNC/$catalog/TableName*
  TableName のストラクチャ定義が返されます。

- *GET /4DSYNC/TableName/FieldName1{,FieldName2},...*
  *TableName*テーブルの*FieldName*フィールドのデータを返します。

- *GET /4DSYNC/TableName/FieldName?$stamp=0&$format=json*
  TableNameテーブルのFieldNameフィールドでstamp 0以降のデータをJSONフォーマットで返します。

- *POST /4DSYNC/TableName/FieldName1{,FieldName2},...*
  クライアントマシン上で行われた変更を4Dデータベースに統合します。

注: データ交換フォーマットにはJavaScript Object Notation (JSON) が使用されます。この書式については別途テクニカルノート等で情報が公開されます。

注: この同期メカニズムを有効にするにはデータベース設定のWeb/設定ページで、**"4DSYNC" URLを許可する**オプションを選択しなければなりません。選択しない場合、4DSYNC URLを含むリクエストは失敗します。

HTTPを使用した同期に関する注意点

/4DSYNC/ URLを使用する場合、以下の原則を適用する必要があります:

- 4Dはデータベースに仮想ストラクチャーが指定されていれば、それを基に動作します。この場合TableName と FieldName 引数はSET FIELD TITLESと[#cmd id="601"/]コマンドで指定されたテーブル名とフィールド名に対応します。これらのコマンドのスコープはセッションであるという点に留意してくだ さい。また4D Serverを使用する場合にはサーバー上のストアドプロシージャーでこれらのコマンドを使用します。
- BLOBおよびピクチャフィールドは、サポートされません。
- データの同期を行うには:
  - HTTPサーバーが起動されていなければなりません。
  - データベース設定のWeb設定ページで、**"4DSYNC" URLを使用したデータベースアクセスを許可**のオプションが選択されていなければなりません (接続セキュリティ のセクションを参照して下さい)。
  - データの同期を行うテーブルごとに、"**複製を有効にする**"プロパティが選択されていなければなりません。仮想ストラクチャーを定義する場合、同期対象のテーブルを仮想ストラクチャーに含めなければなりません。
    警告: このオプションをチェックすることにより、追加の情報が公開されるようになります。データベースへの接続が保護されるよう確認する必要があります (このオプションに関する説明は接続セキュリティを参照)。
- 4Dが**/4DSYNC/**リクエストを受け取ると、On Web Authenticationデータベースメソッド が (存在すれば) 呼び出されます (実際にいつ呼び出されるかは接続セキュリティの図を参照してください)。このデータベースメソッドが**True**を返すとリクエストが実行され、そうでなければ拒否されます。

4D HTMLタグ

4D Webサーバーは、WEB SEND FILEやWEB SEND BLOBコマンドを使用して、Webサーバーから送信されるスタティックなHTMLページに4D変数や式、あるいは様々なタイプの処理への参照を挿入することを可能にする、4D専用のHTMLタグを提供しています。このようなページは**セミダイナミックページ**と呼ばれます。

これらのタグはHTMLコメントとして挿入されます (<!--#Tag Contents-->)。多くのHTMLエディターがコメントの挿入機能を持っています。

以下の4Dタグを使用できます:

- 4DTEXT: 4D変数や4D式を挿入する為に使用します。
- 4DHTML: HTMLコードを挿入するために使用します。
- 4DSCRIPT: 指定された4Dメソッドを実行します。またメソッドの戻り値を挿入します。
- 4DINCLUDE: ページに他のページを挿入するために使用します。
- 4DBASE: 4DINCLUDEで使用されるデフォルトフォルダーを変更します。
- 4DIF, 4DELSE, 4DELSEIF そして 4DENDIF: HTMLコードに条件分岐を挿入するために使用します。
- 4DLOOP と 4DENDLOOP: HTMLコードでループを行うために使用します。

4D HTMLタグについて

4Dが送信するセミダイナミックページの内容解析は、WEB SEND FILE (.htm, .html, .shtm, .shtml) または WEB SEND BLOB (text/html タイプのBLOB) コマンドが呼び出された時、またはWEB SEND TEXTコマンドが呼び出されたとき、さらにはURLを使用してページが呼び出されたときに発生します。
しかし最後のケースでは、最適化の目的で、".htm" と ".html" 拡張子を持つページは解析されません。この場合ページの解析を強制的に行うには、拡張子を ".shtm" または ".shtml" (例えば http://www.server.com/dir/page.shtm) にします。
このタイプのページの例題はWEB GET STATISTICSコマンドの説明にあります。

内部的に、パーサーはUTF-16で動作します。しかし、解析するデータは異なるコード体系かもしれません。タグ(例えば4DHTML)を含むテキストであるときや、文字セットが明示的に指定されているようなときには、4Dは必要に応じてデータを変換します。

4DがHTMLページに含まれるタグを解析して変換する全てのケースを以下の表にまとめます:

動作	**送信ページ解析**	**タグ解析時の文字セット動作**
ページの拡張子 (通常ケース):		
.htm, .html, .shtm, .shtml (HTMLページ)	○	
.xml, .xsl (XMLページ)	○	
.wml (WMLページ)	○	
URLで呼び出されたページ	○, 拡張子が".htm" または ".html" のページを除く	Content-Typeヘッダフィールドの指定に従う。それが無い時はMETA-HTTP EQUIVタグを探す。それも無ければHTTPサーバのデフォルト文字セットを使用。
WEB SEND FILE コマンド呼び出し	○	Content-Typeヘッダフィールドの指定に従う。それが無い時はMETA-HTTP EQUIVタグを探す。それも無ければHTTPサーバのデフォルト文字セットを使用。
WEB SEND TEXT コマンド呼び出し	○	変換は不要
WEB SEND BLOB コマンド呼び出し(BLOBが"text/html"型の場合)	○	Content-Typeヘッダフィールドの指定に従う。それが無い時にはHTTPサーバのデフォルト文字セットを使用。
<!--4DINCLUDE --> タグによる挿入	○	Content-Typeヘッダフィールドの指定に従う。それが無い時はMETA-HTTP EQUIVタグを探す。それも無ければHTTPサーバのデフォルト文字セットを使用。
PROCESS 4D TAGS コマンド呼び出し	○	テキストデータは変換無し。BLOBデータは互換性のためにMac-Romanから自動変換

4Dに処理を行わせるには、HTMLコメントは <!--#4D...--> のフォーマットで記述します。HTMLエディターによっては自動で他の情報をコメントに書き込むことがあることに留意してください。これは動作しないなどの問題を引き起こすことがあります。
しかし他の <!--Beginning of list--> などのHTMLコメントは使用できます。

コメント <!--#4D... が -->で終了していない場合、メッセージ "<!--#4D... : --> が必要です" が挿入され、このレベルで解析が停止します (エラーを示すため、ページが送信される)。
複数のタイプのコメントをミックスできます。例えば以下のHTMLシンタックスが可能です:

```
<HTML>
...
<BODY>
<!--#4DSCRIPT/PRE_PROCESS-->    (メソッド呼び出し)
<!--#4DIF myvar=1-->    (If 条件)
   <!--#4DINCLUDE banner1.html-->    (サブページ挿入)
<!--#4DENDIF-->    (End if)
<!--#4DIF myvar=2-->
   <!--#4DINCLUDE banner2.html-->
<!--#4DENDIF-->

<!--#4DLOOP [テーブル]-->     (カレントセレクションに基づくループ)
<!--#4DIF [テーブル]ValNum>10-->    (If [テーブル]ValNum>10)
   <!--#4DINCLUDE subpage.html-->    (サブページ挿入)
<!--#4DELSE-->    (Else)
   <B>Value: <!--#4DTEXT [テーブル]ValNum--></B><BR>
      (フィールド表示)
<!--#4DENDIF-->
<!--#4DENDLOOP-->    (End for)
</BODY>
</HTML>
```

4DTEXT

シンタックス: <!--#4DTEXT VarName--> または <!--#4DTEXT 4DExpression-->

タグ <!--#4DTEXT VarName--> を使用して4D変数や式への参照 VarName をHTMLページの任意の場所に挿入します。例えば以下のように記述すると:

<P>Welcome to <!--#4DTEXT vtSiteName-->!</P>

4D変数 vtSiteName の値がHTMLページに送信時に挿入されます。値はテキストとして挿入されます。

4DTEXT タグを使用して、変数だけでなく4D式も挿入できます。フィールドの値を直接挿入できますし (例 <!--#4DTEXT [tableName]fieldName-->)、または配列要素の値も挿入できますし (例 <!--#4DTEXT tabarr{1}-->)、値を返すメソッドも使用できます (<!--#4DTEXT mymethod-->)。

式の変換は変数のそれと同じルールが適用されます。さらに式は4Dのシンタックスルールに適合していなければなりません。

注: 4DTEXT での4Dメソッドの実行は、メソッドプロパティの"4DHTMLタグとURL (4DACTION) で利用可能"属性の設定に基づきます。詳細は 接続セキュリティ を参照してください。

式に4D関数への呼び出しを直接含めることができますが、これはローカライズの観点から推奨されません。例えば <!--#4DTEXT Current date--> は英語バージョンの4Dでは正しく解釈されますが、フランス語バージョンの4Dでは解釈されません。同じことが実数値にも言えます (小数点は言語により異なります)。どちらの場合にもプログラムを使用して変数値を割り当てることを強くお勧めします。

解釈エラーの場合、"<!--#4DTEXT myvar--> : ## error # error code" のように表示されます。

注:

- プロセス変数を使用できます。
- ピクチャーフィールドの内容を表示できます。しかしピクチャー配列の項目内容を表示することはできません。
- HTMLはワープロ志向のアプリケーションなので、通常はテキスト変数を使用します。しかしBLOB変数を使用することもできます。この場合長さ情報なしのテキストBLOBを使用します。

互換性に関する注意: 4D v12.2より、4DVARタグは4DTEXTタグに置き換えられます。4DVARタグは以前のバージョンと同様に動作しますが、テキストの挿入には 4DTEXTタグを使用することを強く推奨します。既存のプロジェクトもすべて4DTEXTに置き換えてください。

4DHTML

シンタックス: <!--#4DHTML VarName--> または <!--#4DHTML 4DExpression-->

この4DHTMLタグを使用すると、変数や4D式の値をHTMLソースとしてページに挿入できます。4DTEXTと異なる点は値がHTMLソースとして挿入されることです。

例えば4Dのテキスト変数myVarを4Dタグを使用して挿入した結果は以下の様になります:

myvar 値	**タグ**	**Webページに挿入されるテキスト**
myvar:="<B>"	<!--#4DTEXT myvar-->	&lt;B&gt;
myvar:="<B>"	<!--#4DHTML myvar-->	<B>

解釈エラーの場合、"<!--#4DHTML myvar--> : ## エラー # エラーコード"のように表示されます。

注: 4DHTMLでの4Dメソッドの実行は、メソッドプロパティの"4D HTMLタグおよびURL (4DACTION) で利用可能"属性の設定に基づきます。詳細は 接続セキュリティ を参照してください。

互換性に関する注意: 4D v12.2より、4DHTMLVARタグは4DHTMLタグに置き換えられます。4DHTMLVARタグは以前のバージョンと同様に動作しますが、可読性向上のために、HTMLコード挿入には4DHTMLタグを使用することを推奨します。

4DSCRIPT/

シンタックス: <!--#4DSCRIPT/MethodName/MyParam-->

4DSCRIPT タグを使用して、スタティックなHTMLページを送信する際に4Dメソッドを実行することを可能にします。<!--#4DSCRIPT/MyMethod/MyParam--> タグがHTMLコメントとしてページに現れると、MyMethodメソッドが$1引数にMyParamを受け取って実行されます。Webページをロードする際、4Dは On Web Authenticationデータベースメソッド を (存在すれば) 呼び出します。このメソッドが**True**を返すと、4Dはメソッドを実行します。
メソッドは$0にテキストを返します。文字列がコード1から始まっていると、それは　HTMLソースとして扱われます (変数のときと同じ原則)。

注: 4DSCRIPTでの4Dメソッドの実行は、メソッドプロパティの"4D HTMLタグおよびURL (4DACTION) で利用可能"属性の設定に基づきます。詳細は 接続セキュリティ を参照してください。

ページ内容の解析は WEB SEND FILE (.htm, .html, .shtm, .shtml)、または WEB SEND BLOB (text/htmlタイプのBLOB) が呼び出されたときに行われます。またURLが".shtm" や ".shtml"拡張子のファイルを指しているときにも解析が行われます (例えば http://www.server.com/dir/page.shtm)。

例えば、以下のコメントをスタティックページに挿入したとしましょう "Today is <!--#4DSCRIPT/MYMETH/MYPARAM-->" 。ページをロードする際、4Dは On Web Authenticationデータベースメソッド を (存在すれば) 呼び出し、そして MYMETH メソッドの$1引数に文字列 "/MYPARAM" を渡して呼び出します。

メソッドは$0にテキストを返します (例えば "10/12/31")。コメント "Today is <!--#4DSCRIPT/MYMETH/MYPARAM-->" は結果 "Today is 10/12/31" となります。

MYMETHメソッドは以下のとおりです:

```
 C_TEXT($0)\\This parameter must always be declared
 C_TEXT($1)\\This parameter must always be declared
 $0:=String(Current date)
```

警告: $0 と $1 引数を宣言しなければなりません。

注: 4DSCRIPTから呼び出されるメソッドはインタフェース要素 (DIALOG, ALERT...) を呼び出してはいけません。

4Dはメソッドを見つけた順に実行するので、ドキュメント中で離れて参照される変数の値を設定するメソッドを呼び出すことも可能です。モードは関係ありません。

注: スタティックページに必要なだけ <!--#4DSCRIPT...--> コメントを記述できます。

4DINCLUDE

シンタックス: <!--#4DINCLUDE Path-->

このコメントを使用して、(path引数で指定された) 他のHTMLページを、これから送信するHTMLに含めることができます。デフォルトでHTMLのボディー部、つまり<body> と</body>タグの間の内容だけが統合されます (bodyタグは含められません)。これによりヘッダーに含まれるメタタグ関連の衝突が回避されます。しかしpathで指定されたHTML中に<body></body>タグが含まれない場合、ページ全体が統合されます。この場合メタタグの整合性を管理するのは開発者の役割です。

<!--#4DINCLUDE --> コメントは、テスト (<!--#4DIF-->) やループ (<!--#4DLOOP-->) と使用するととても便利です。条件に基づきあるいはランダムにタグを含める便利な方法です。
このタグを使用してページをインクルードするとき、拡張子にかかわらず、4Dは呼び出されたページを解析してから、内容を4DINCLUDE呼び出し元のページに挿入します。

<!--#4DINCLUDE --> コメントで挿入されたページはロードされ、URLで呼ばれてWEB SEND FILEコマンドで送信されたファイルと同じように、Webサーバーキャッシュに格納されます。

pathには、含めるドキュメントへのパスを記述します。

警告: 4DINCLUDE呼び出しの場合、パスは解析される親ドキュメントからの相対パスです。フォルダ区切り文字にはスラッシュ (/) を使用し、レベルをさかのぼるには2つのドットを使用します (HTMLシンタックス)。

注:

- PROCESS 4D TAGSコマンドで4DINCLUDEタグを使用する場合、デフォルトフォルダはデータベースストラクチャを含むフォルダーです。
- 4DINCLUDE タグで使用されるデフォルトフォルダーを<!--#4DBASE -->タグを使用して変更できます (後述)。

ページ内の<!--#4DINCLUDE path-->数に制限はありません。しかし<!--#4DINCLUDE path-->の呼び出しは1レベルのみ有効です。つまり、例えばmydoc1.html内の<!--#4DINCLUDE mydoc2.html-->で呼ばれるmydoc2.htmlページのボディに、<!--#4DINCLUDE mydoc3.html--> を含めることはできません。

さらに、4Dは組み込み呼び出しが再帰的でないか確認します。

エラーの場合、"<!--#4DINCLUDE path--> : ドキュメントを開けません"のように表示されます。

例題

```
<!--#4DINCLUDE subpage.html-->
<!--#4DINCLUDE folder/subpage.html-->
<!--#4DINCLUDE ../folder/subpage.html-->
```

## 4DBASE

シンタックス: <!--#4DBASE folderPath-->

<!--#4DBASE --> タグは<!--#4DINCLUDE-->タグで使用されるワーキングディレクトリを指定します。
Webページ内でこのタグが使用されると、<!--#4DBASE --> タグはこのページ内でそのあとに続くすべての<!--#4DINCLUDE--> 呼び出しのディレクトリを変更します。組み込まれたファイル内で<!--#4DBASE -->フォルダーが変更されると、親のファイルから元となる値を取得します。

folderPath 引数には現在のページに対する相対パスを指定し、パスは"/"で終わっていなければなりません。指定するフォルダーはWebフォルダー内になければなりません。
WEBFOLDER キーワードを渡すと、デフォルトパスに戻されます (そのページに対して相対)。

4D v12では以下のように各呼び出しごとに相対パスをしていなければなりませんでした:

```
<!--#4DINCLUDE subpage.html-->
<!--#4DINCLUDE folder/subpage1.html-->
<!--#4DINCLUDE folder/subpage2.html-->
<!--#4DINCLUDE folder/subpage3.html-->
<!--#4DINCLUDE ../folder/subpage.html-->
```

<!--#4DBASE --> タグを使用すれば以下のように記述できます:

```
<!--#4DINCLUDE subpage.html-->
<!--#4DBASE folder/-->
<!--#4DINCLUDE subpage1.html-->
<!--#4DINCLUDE subpage2.html-->
<!--#4DINCLUDE subpage3.html-->
<!--#4DBASE ../folder/-->
<!--#4DINCLUDE subpage.html-->
<!--#4DBASE WEBFOLDER-->
```

Example

<!--#4DBASE --> タグを使用してホームページのディレクトリを設定する:

```
/* Index.html */
<!--#4DIF LangFR=True-->
    <!--#4DBASE FR/-->
<!--#4DELSE-->
    <!--#4DBASE US/-->
<!--#4DENDIF-->
<!--#4DINCLUDE head.html-->
<!--#4DINCLUDE body.html-->
<!--#4DINCLUDE footer.html-->
```

head.html ファイル内でカレントフォルダーが*<!--#4DBASE -->を使用して変更されているが*、index.html内では変更されない:

```
/* Head.htm */
/* ここでのワーキングディレクトリはインクルードされるファイルに対して相対的 (FR/ または US/) */
<!--#4DBASE Styles/-->
<!--#4DINCLUDE main.css-->
<!--#4DINCLUDE product.css-->
<!--#4DBASE Scripts/-->
<!--#4DINCLUDE main.js-->
<!--#4DINCLUDE product.js-->
```

4DIF, 4DELSE, 4DELSEIF and 4DENDIF

シンタックス: <!--#4DIF expression--> {<!--#4DELSEIF expression2-->...<!--#4DELSEIF expressionN-->} {<!--#4DELSE-->} <!--#4DENDIF-->

<!--#4DELSEIF--> (オプション), <!--#4DELSE--> (オプション) と <!--#4DENDIF--> コメントと共に使用することで、<!--#4DIF expression--> コメントはHTMLコードに条件分岐を実行させることを可能にします。

expression 引数はブール値を返す有効な4D式です。式は括弧の中に記述され、4Dのシンタックスルールに準拠していなければなりません。

<!--#4DIF expression--> ... <!--#4DENDIF--> を複数レベルでネストできます。4Dのようにそれぞれの <!--#4DIF expression--> は <!--#4DENDIF-->とマッチしなければなりません。

解釈エラーの場合、<!--#4DIF --> と <!--#4DENDIF-->"の間のコンテンツの代わりに、<!--#4DIF expression-->: ブール式が必要です"が挿入されます。

同様に、<!--#4DIF -->が同じ数の<!--#4DENDIF-->で閉じられていない場合、"<!--#4DIF expression-->: 4DENDIFが必要です" が<!--#4DIF --> と <!--#4DENDIF-->の間のコンテンツの代わりに挿入されます。

<!--#4DELSEIF--> タグを使用すると、条件テストの記述が容易になります。最初に**True**と判定されたブロック内にあるコードだけが実行されます。Trueブロックがない場合、文は実行されません (<!--#4DELSE-->がなければ)。
最後の<!--#4DELSEIF-->の後に<!--#4DELSE-->タグを記述できます。すべての条件が**False**の場合、<!--#4DELSE-->ブロックの文が実行されます。

以下の2つのコードは同等です

- 4DELSEのみを使用する場合:

  ```
  <!--#4DIF Condition1-->
    /* Condition1 is true*/
  <!--#4DELSE-->
      <!--#4DIF Condition2-->
          /* Condition2 is true*/
      <!--#4DELSE-->
          <!--#4DIF Condition3-->
              /* Condition3 is true */
          <!--#4DELSE-->
              /*None of the conditions are true*/
          <!--#4DENDIF-->
      <!--#4DENDIF-->
  <!--#4DENDIF-->
  ```

- 同じ内容を4DELSEIFタグを使用して記述した場合:

  ```
  <!--#4DIF Condition1-->
       /* Condition1 is true*/
  <!--#4DELSEIF Condition2-->
       /* Condition2 is true*/
  <!--#4DELSEIF Condition3-->
      /* Condition3 is true */
  <!--#4DELSE-->
      /* None of the conditions are true*/
  <!--#4DENDIF-->
  ```

例題 1

スタティックなHTMLページに書かれたこの例題のコードは、vname#"" 式の結果に応じ、異なるラベルを表示します:

```
<BODY>
...
<!--#4DIF vname#""-->
Names starting with <!--#4DVAR vname-->.
<!--#4DELSE-->
No name has been found.
<!--#4DENDIF-->
...
</BODY>
```

例題 2

この例題は接続したユーザーに基づき異なるページを返します:

```
<!--#4DIF LoggedIn=False-->
    <!--#4DINCLUDE Login.htm -->
<!--#4DELSEIF User="Admin"-->
    <!--#4DINCLUDE AdminPanel.htm -->
<!--#4DELSEIF User="Manager"-->
    <!--#4DINCLUDE SalesDashboard.htm -->
<!--#4DELSE-->
    <!--#4DINCLUDE ItemList.htm -->
<!--#4DENDIF-->
```

4DLOOP と 4DENDLOOP

シンタックス: <!--#4DLOOP condition--> <!--#4DENDLOOP-->

このコメントを使用して、条件を満たす間、HTMLコードの一部を繰り返すことができます。繰り返し部のコードは<!--#4DLOOP--> と <!--#4DENDLOOP-->で挟まれます。

<!--#4DLOOP condition--> ... <!--#4DENDLOOP--> ブロックはネストできます。4Dと同様、それぞれの<!--#4DLOOP condition--> は同じ数の<!--#4DENDLOOP-->で閉じられていなければなりません。

条件には3種類あります:

<!--#4DLOOP [テーブル]-->
このシンタックスは、指定されたテーブルのカレントプロセスのカレントセレクションに基づき、レコードごとにループします。2つのコメントの間のHTMLコードはカレントセレクションレコード毎に繰り返されます。

注: 4DLOOPタグがテーブルを条件として使用されると、レコードが読み込みのみでロードされます。

以下のHTMLコードは:

```
<!--#4DLOOP [People]-->
<!--#4DTEXT [People]Name--> <!--#4DTEXT [People]Surname--><BR>
<!--#4DENDLOOP-->
```

4Dランゲージで表すと以下のとおりです:

```
 FIRST RECORD([People])
 While(Not(End selection([People])))
  // ...
    NEXT RECORD([People])
 End while
```

<!--#4DLOOP 配列-->
このシンタックスは、配列項目ごとにループします。2つのコメントの間のHTMLコードが繰り返されるたびに、配列のカレント項目がインクリメントされます。

注: このシンタックスで二次元配列を使用することはできません。この場合、ネストしたループでメソッドを条件に使用します。

以下のHTMLコードは:

```
<!--#4DLOOP arr_names-->
<!--#4DTEXT arr_names{arr_names}--><BR>
<!--#4DENDLOOP-->
```

4Dランゲージで表すと以下のとおりです:

```
 For($Elem;1;Size of array(arr_names))
    arr_names:=$Elem
  //...
 End for
```

<!--#4DLOOP method-->
この シンタックスでは、メソッドがTrueを返す間ループが行われます。メソッドは倍長整数タイプの引数を受け取ります。まずメソッドは引数0を渡されます。これは (必要に応じて) 初期化ステージとして使用できます。その後、Trueが返されるまで1, 2, 3と渡される引数値がインクリメントされます。

セキュリティのため、On Web Authenticationデータベースメソッドが初期化ステージ (引数に0が渡されて実行される) の前に一度呼び出されます。認証されると、初期化に進みます。

警告: コンパイルのため、C_BOOLEAN($0) とC_LONGINT($1) が宣言されていなければなりません。

例題 以下のHTMLコードは:

```
<!--#4DLOOP my_method-->
<!--#4DTEXT var--> <BR>
<!--#4DENDLOOP-->
```

4Dランゲージで表すと以下のとおりです:

```
 If(AuthenticationWebOK)
    If(my method(0))
```

```
        --
       $counter:=1
       While(my_method($counter))
  //...
          $counter:=$counter+1
       End while
    End if
 End if
```

my_method は以下のようになります:

```
 C_LONGINT($1)
 C_BOOLEAN($0)
 If($1=0)
  `Initialisation
    $0:=True
 Else
    If($1<50)
  //...
       var:=...
       $0:=True
    Else
       $0:=False `Stops the loop
    End if
 End if
```

解釈エラーの場合、"<!--#4DLOOP expression-->: エラーの説明" が<!--#4DLOOP --> と <!--#4DENDLOOP-->の間のコンテンツの代わりに挿入されます。

以下のメッセージが表示されます:

- 予期しない式のタイプ (標準のエラー);
- テーブル名が正しくありません (テーブル名のエラー);
- 配列が必要です (変数が配列でないか、二次元配列が指定された);
- メソッドが存在しません;
- シンタックスエラー (メソッド実行時);
- アクセス権エラー (テーブルやメソッドにアクセスする権限がない)。
- 4DENDLOOP が必要です (<!--#4DLOOP -->が対応する<!--#4DENDLOOP-->で閉じられていない)。

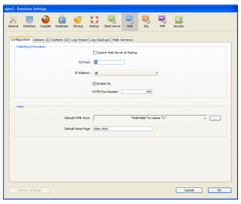
Webサーバー設定

データベース設定の**Web**ページで定義されたパラメタを使用して、4D Webサーバの動作を設定できます。この節ではこのページの**設定**、**オプション (I)** と **(II)**について説明します。

- **ログ**ページはWebサイトに関する情報で説明しています。.
- **Webサービス**ページは"Design Reference"マニュアルで説明しています。

互換性に関する注意: 以前のバージョンの4Dにある特定のWebメカニズムは廃止予定となっています (例えば未知のURLから"/"を取り除かない)。互換性の目的で、これらのメカニズムは変換されたデータベースで利用可能です。この場合、データベース設定の**互換性**ページで必要に応じて無効にできます。

設定ページ


new2 - Database Settings

Publishing Information

☐ Launch Web Server at Startup

TCP Port: 80

IP Address: All

☑ Enable SSL

HTTPS Port Number: 443

Paths

Default HTML Root: "WebFolder" in volume "C:"

Default Home Page: index.html

起動時にWebサーバを開始する

4Dアプリケーションの起動時にWebサーバを起動するか指定します。このオプションはWebサーバ設定と接続管理で説明しています。

TCPポート番号

デフォルトで、4Dは通常のWeb TCPポート番号である80番を使用してWebデータベースを公開します。他のWebアプリケーションによってこのポート番号が既に使用されている場合、4Dが使用するポート番号を変更する必要があります。Mac OS Xでは、TCPポート番号を変更することで、rootユーザを有効にしなくてもWebサーバを開始することができます。(Webサーバ設定と接続管理参照)
ポート番号を変更するには、**TCPポート**入力エリアに適切な値 (同じマシン上で動作する他のTCP/IPサービスが使用していないTCPポート番号)を指定します。

Note: 0を指定すると、4DはデフォルトのTCPポート番号80を使用します。

デフォルトでないTCPポート番号を使用して公開されたWebサーバに接続するには、Webブラウザに入力するアドレスに使用するポート番号を含めなければなりません。アドレスの後にコロンに続けてポート番号を指定します。例えばTCP番号8080を使用する場合、"123.4.567.89:8080"のように書きます。

警告: デフォルトのTCPポート番号 (標準モードで80、SSLモードで443) 以外を使用する場合、同時に使用する他のサービスのデフォルトポート番号を使用しないよう注意してください。例えば、WebサーバマシンでFTPプロトコルを使用する計画である場合、(あなたが何を行おうとしているか分かっていないのであれば) このデフォルトポートであるTCPポート20と21は使用してはいけません。標準のTCPポート番号割り当てを知るには、4D Internet CommandsのAppendix B - TCPポート番号の節を参照してください。256より下のポート番号はwell-knownサービスに予約されています。256から1024はUNIXプラットフォーム由来のサービスに予約されています。互換性のためにこれらの数値よりも上、例えば2000台や3000台などを指定します。

HTTPリクエストを受け付けるIPアドレスの指定

WebサーバがHTTPリクエストを受け付けるIPアドレスを指定できます。

注: 4D v14 以降、データベース設定の"IPアドレス" リストにて**すべて**が選択されていた場合は、HTTP サーバーは自動的にIPv6アドレスをサポートします。詳細な情報に関しては、Support of IPv6を参照して下さい。

デフォルトで、サーバはすべてのIPアドレスへのリクエストに応答します (**すべて**オプション)。

ドロップダウンリストにはマシンで利用可能なIPアドレスがリストされます。特定のアドレスを指定すると、サーバはこのIPアドレスへのリクエストにのみ応答します。

この機能は複数のIPアドレスが設定されたマシン上で動作する4D Webサーバのためのものです。これはしばしばインターネットホストプロバイダで使用されます。

このようなマルチホーミングシステムの実装はWebサーバマシン上での特定の設定を必要とします:

Mac OSにセカンダリのIPアドレスをインストールする

Mac OS上でマルチホーミングを設定するには:

1. **TCP/IP** コントロールパネルを開きます。
2. **設定** ポップアップメニューから**手動**オプションを選択します。
3. テキストファイルを"Secondary IP Addresses"という名称で作成し、SystemフォルダのPreferencesサブフォルダに保存します。
4. "Secondary IP Addresses"ファイルの各行にセカンダリIPアドレスと、オプションでサブネットマスクおよびセカンダリIPアドレス用のルータアドレスを記述します。

詳細はApple社のドキュメントを参照してください。

WindowsにセカンダリのIPアドレスをインストールする

Windows上でマルチホーミングを設定するには:

1. 以下のコマンド (またはWindows OSごとの同様の機能) を選択します:
   スタート メニュー > **コントロールパネル** > **ネットワークとインターネット接続** > **ネットワーク接続** > **ローカルエリア接続** (プロパティ) > **インターネットプロトコル (TCP/IP)** > **プロパティ** ボタン > **詳細...** ボタン。"詳細 TCP/IP 設定"ダイアログが表示されます。
2. "IPアドレス"エリアの**追加...** ボタンをクリックし、IPアドレスを追加します。

最大5つまでIPアドレスを定義できます。これを行うにはシステム管理者に問い合わせる必要があります。詳細はWindowsのドキュメントを参照してください。

SSLを有効にする

Webサーバがセキュアな接続を受け入れるか受け入れないかを指定します。このオプションはSSLプロトコルの使用で説明します。

HTTPSポート番号

SSL (HTTPSプロトコル) を使用したセキュアなHTTP接続に対してWebサーバが使用するTCPポート番号を指定できます。デフォルトでHTTPSポート番号は443です。
この番号の変更を検討する主な理由は2つあります:

- セキュリティ: 攻撃者は標準のTCPポート (80 and 443) に集中して攻撃を仕掛けてくるかもしれません。
- Mac OS Xでは、特定のポート番号 (0 から 1023) を使用してWebサーバを起動するには、特別なアクセス権を必要とします。rootユーザだけがこれらのポートを使用するアプリケーションを起動できます。標準ユーザがWebサーバを起動するには、TCPポート番号の変更が一つのソリューションとなります (Webサーバ設定と接続管理 の節参照)。
  有効であれば何番でも渡すことができます (Mac OS X 上でのアクセス上の制限を避けるためには、1023より上の番号を指定します)。TCPポート番号に関する詳細は、先述の "TCPポート番号" を参照してください。

4DSYNC URLを使用したアクセスを許可する

このオプションを使用して/4DSYNC URLによるHTTP同期サポートを制御します。この機能については接続セキュリティで説明されています。

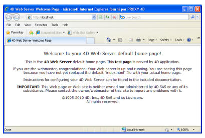
デフォルトHTMLルート

Webサイトファイルのデフォルトの位置を指定し、ディスク上でその場所より階層的に上のファイルにアクセスできない場所を指定します。このオプションは接続セキュリティの節で説明しています。

デフォルトホームページの設定

データベースに接続するすべてのブラウザー用にデフォルトホームページを指定できます。このページはスタティックでもセミダイナミックでも可能です。

デフォルトでWebサーバーが最初に起動されたとき、4Dは"index.html"という名前のホームページを作成し、HTMLルートフォルダーに置きます。この設定を変更しない場合、Webサーバーに接続するブラウザーには以下のようなページが表示されます:


4D Web Server Welcome Page - Microsoft Internet Explorer fourni par PROXY 4D

http://localhost/

Welcome to your 4D Web Server default home page!

This is the **4D Web Server** default home page. This **test page** is served by 4D Application.

If you are the webmaster, congratulations! Your Web server is up and running. You are seeing this page because you have not yet replaced the default "index.html" file with your actual home page.

Instructions for configuring your 4D Web Server can be found in the included documentation.

IMPORTANT: This Web page or Web site is neither owned nor administered by 4D SAS or any of its subsidiaries. Please contact the owner/webmaster of this site to report any problems with it.

©1995-2010 4D, Inc., 4D SAS and its Licensors.
All rights reserved.

デフォルトホームページを変更するには、ルートフォルダーの"index.html"ページを置き換えるか、使用するページのアクセス相対パスを"デフォルトホームページ"エリアに入力します。
アクセスパスはデフォルトHTMLルートからの相対パスで設定しなければなりません。
データベースのマルチプラットフォーム互換性を確保するために、4D Webサーバーは特別なアクセスパスの記法を使用します。シンタックスルールは以下のとおりです:

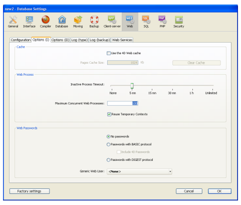
- フォルダーはスラッシュ ("/") で区切ります。
- アクセスパスはスラッシュ ("/") で終わっていてはなりません。
- フォルダー階層を1レベル上がるには、フォルダー名の前に "../" を記述します。
- アクセスパスはスラッシュ ("/") で始まっていてはいけません。

例えばデフォルトホームページを、デフォルトHTMLルートフォルダー内のWebサブフォルダーにある"MyHome.htm"にする場合、"Web/MyHome.htm"と入力します。

注: WEB SET HOME PAGEコマンドを使用して、Webプロセスごとにデフォルトホームページを設定できます。

デフォルトのカスタムホームページを指定しない場合、On Web Connectionデータベースメソッドが呼び出されます。プロシージャーでリクエストを処理するのは開発者の役割です。

オプション (I) ページ


new2 - Database Settings

Cache

☐ Use the 4D Web cache

Pages Cache Size: 1024 Kb

Web Process

Inactive Process Timeout: 5 mn

Maximum Concurrent Web Processes: 100

☑ Reuse Temporary Contexts

Web Passwords

● No passwords

○ Passwords with BASIC protocol

☐ Include 4D Passwords

○ Passwords with DIGEST protocol

Generic Web User: <None>

スタティックページのキャッシュ

4D Webにはキャッシュがあり、スタティックページ, GIF, JPEG (<128 kb) そしてスタイルシート (.css files) などがリクエストされると、メモリにロードされます。
キャッシュの利用は、スタティックページの送信時にWebサーバのパフォーマンスを劇的に向上します。

キャッシュはすべてのWebプロセスで共有されます。環境設定でキャッシュのサイズを設定できます。デフォルトでスタティックページのキャッシュは無効になっています。有効にするには、**4D Webキャッシュを使用する**オプションをチェックします。

ページキャッシュサイズエリアでキャッシュのサイズを変更できます。この設定はスタティックページのサイズや数、およびホストマシンで利用可能なリソースによります。

注: Webデータベースを利用する間、WEB GET STATISTICSコマンドを使用してキャッシュのパフォーマンスを検証できます。例えばキャッシュ利用率が100%,に近い場合、キャッシュに割り当てたメモリ量を増やすことを考慮します。
/4DSTATS と /4DHTMLSTATS URLも、キャッシュの状態を知るために使用できます。Webサイトに関する情報を参照してください。

キャッシュが有効になると、4D Webサーバはブラウザからリクエストされたページをまずキャッシュから探します。ページが見つかると、即座にそれを送信します。見つからなければ、4Dはディスクからページをロードし、それをキャッシュに置きます。
キャッシュがいっぱいで、新しいページをキャッシュに置く必要がある場合、4Dは最も使われていないページの中から、もっとも古いページをアンロードします。

キャッシュのクリア

いつでもページやイメージをキャッシュからクリアできます (例えばスタティックページを更新し、キャッシュにそれをリロードさせたい場合)。
これを行うには、**キャッシュクリア** ボタンをクリックします。キャッシュは即座にクリアされます。

非動作プロセスのタイムアウト

サーバー上で、活動していないWebプロセスを閉じるための最大タイムアウト時間を設定できます。

最大同時Webプロセス

このオプションは、サーバー上で同時に開くことのできるすべてのWebプロセスの最大同時接続数を設定します(標準Webプロセス、またはプールされたプロセスなどすべて)。このパラメターは、異常な数のリクエストによる4D Webサーバの飽和状態を避けるために使用します。

デフォルトでこの値は32000です。10から32000の値を設定できます。

最大Web同時接続数 (マイナス1) に達すると、4Dは新しいプロセスを作成せず、"Server unavailable" (ステータス HTTP 503 - Service Unavailable) を返信します。

注: Webの最大接続数はWEB SET OPTIONコマンドで設定することもできます。

プロセスのプールについて

WebプロセスのプールはWebサーバーの応答性能を向上させます。リサイクルするプロセス数は最小値 (デフォルトで0) から最大値 (デフォルトで10) の間です。この数はSET DATABASE PARAMETERコマンドを使用して変更できます。Webプロセス最大同時接続数が変更されると、プール数の最大値が最大同時接続数を超えている場合は、最大同時接続数まで下げられます。

正しい値を決定する

理論上は、Webプロセスの最大数は以下の式で求められます: 利用可能なメモリ/Webプロセススタックサイズ
他のソリューションはランタイムエクスプローラでWebプロセスを監視することです。Webプロセスの現在数と、Webサーバが実行されてからの最大数が示されます。

自動セッション管理

4D HTTPサーバーによるユーザーセッションの自動管理内部メカニズムを有効にしたり無効にしたりします。このメカニズムについてはWebセッション管理で説明しています。

4D v13以降に作成されたデータベースではデフォルトでこのメカニズムが有効になっています。他方互換性のため、v12以前のバージョンから変換されたデータベースでは無効になっています。この機能を利用するためには明示的に有効にしなければなりません。

このオプションが選択されていると、"一時的なコンテキストを再利用する"オプションも自動で選択され、ロックされます。

一時的なコンテキストを再利用する (リモートモード) (in remote mode)

前のWebリクエストを処理するために作成されたWebプロセスを再利用することによって、4Dリモートモードで実行されているWebサーバーの動作を最適化できます。実際、4D WebサーバーはそれぞれのWebリクエストを処理するために専用のWebプロセスを必要とします。リモートモードでは必要に応じて、このプロセスはデータやデータベースエンジンにアクセスするために4D Serverに接続し増す。そしてプロセス独自の変数やセレクションを使用してコンテキストを作成します。リクエストの処理が終了すると、このプロセスは廃棄されます。

一時的なコンテキストを再利用するオプションがチェックされていると、リモートモードの4Dは作成された固有のWebプロセスを保守し、その後のリクエストで再利用します。プロセスの作成処理が省略されるため、Webサーバーのパフォーマンスが向上します。

他方このオプションを使用する場合、不正な結果が返されることを避けるために、4Dメソッド内で使用される変数をシステマチックに初期化するようにしてくだ さい。同様に、以前のリクエストで使用されたカレントセレクションやカレントレコードをアンロードする必要があります。

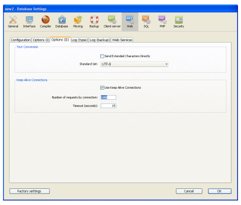
注:

- **自動セッション管理**が有効にされると、このオプションも選択されロックされます。実際セッション管理メカニズムはWebプロセスの再利用の原則に基づいています: 各Webセッションはセッションの寿命が有効である間、同じWebプロセスを使用して処理されます。異なるセッションでWebプロセスが共有されることはありません。セッションが終了するとプロセスは破棄され、再利用されることはありません。なのでこの場合セレクションや変数を初期化する必要はありません。
- このオプションはリモートモードの4D Webサーバーでのみ効果があります。ローカルモードの4Dでは (セッション管理を行うプロセスを除く) すべてのWebプロセスが使用後に終了されます。

Webパスワード

パスワードを使用したWebサイトアクセス保護の設定を行います。このオプションはの節で説明しています。

オプション (II) ページ


new2 - Database Settings

Text Conversion

☐ Send Extended Characters Directly

Standard Set: UTF-8

Keep-Alive Connections

☑ Use Keep-Alive Connections

Number of requests by connection: 100

Timeout (seconds): 15

拡張文字をそのまま送信

デフォルトで、4D Webサーバはダイナミックおよびスタティックページの拡張文字を、HTML標準に基づき送信前に変換します。このようなページはブラウザで評価されます。
拡張文字を変換せずそのまま送信するようWebサーバを設定できます。このオプションにより、特にShift_JIS文字コード利用時の日本語のシステムで速度が向上します。

この設定を行うには、**拡張文字をそのまま送信** オプションをチェックします。

スタンダードセット

スタンダードセットドロップダウンリストを使用して、4D Webサーバが使用する文字セットを定義できます。デフォルトで文字セットはUTF-8です。4DのWebサーバはKeep-Alive接続を使用できます。Keep-Aliveオプションを使用して、一つの開かれたTCP接続でWebブラウザとサーバ間の一連のデータ交換を保守し、システムリソースを節約して、通信を最適化できます。

Keep-Alive接続

Keep-Alive接続を使用するオプションは、WebサーバのKeep-Alive接続を有効および無効にします。このオプションはデフォルトで有効になっています。ほとんどの場合、通信が高速化されるため、この状態をお勧めします。WebブラウザがKeep-Alive接続をサ ポートしない場合、4D Webサーバは自動でHTTP/1.0にスイッチします。
4D WebサーバのKeep-Alive機能はすべてのTCP/IP接続 (HTTP, HTTPS) に関連します。しかしながらすべての4D Webプロセスで常にKeep-Alive接続が使用されるわけではないことに留意してください。あるケースでは、内部的な他の最適化機能が呼び出される ことがあります。Keep-Alive接続は特にスタティックページで有効です。

Keep-Alive接続を設定する2つのオプションがあります:

- **接続毎のリクエスト数**: ひとつのKeep-Alive接続におけるリクエストとレスポンスの最大数を設定します。接続あたりのリクエスト数を制限することで、サーバのリクエスト過多を避けることができます (攻撃者が使用するテクニック)。
  4D Webサーバをホストするマシンのリソースに応じて、デフォルト値 (100) を増減できます。
- **タイムアウト**: この値を使用して、Webブラウザからのリクエストが行われない状態で、Webサーバが開かれた接続を保守する最大の待ち秒数を設定します。この秒数が経過すると、サーバは接続を閉じます。

接続が閉じられた後にWebブラウザがリクエストを送信すると、新しいTCP接続が作成されます。この動作はユーザからは見えません。

Webサイトに関する情報

4Dは4D Webサイトの機能に関する情報を提供します。

- 特定のURLを使用してサイトをコントロールできます (/4DSTATS、/4DHTMLSTATS、/4DCACHECLEARと /4DWEBTEST) 。
- すべてのリクエストのログを作成できます。
- ランタイムエクスプローラウィンドウのウオッチページにあるWeb Serverに関する情報を取得できます。

Web Server管理用のURL

4D Web Serverは、/4DSTATS、/4DHTMLSTATS、/4DCACHECLEARと /4DWEBTEST の4つのURLを受け入れます。

- /4DSTATS、/4DHTMLSTATS と /4DCACHECLEAR はデータベースの設計者と管理者のみが利用可能です。データベースの4Dパスワードシステムが起動されていないと、これらのURLはすべてのユーザに対して利用可能となります。
- /4DWEBTEST は、常に利用可能です。

/4DSTATS

/4DSTATS URLは以下の情報をHTMLの表形式で返します。:

- **現在のキャッシュサイズ**: Webサーバーの現在のキャッシュサイズ (バイト)
- **最大キャッシュサイズ**: キャッシュの最大サイズ (バイト)
- **キャッシュされたオブジェクトの最大サイズ**: キャッシュされたオブジェクト中で最も大きなもの (バイト)
- **使用キャッシュ**: 使用されているキャッシュの率
- **キャッシュされているオブジェクト**: キャッシュされているオブジェクトの数 (ページやピクチャーファイル等)。

(*) スタティックページとピクチャのキャッシュに関する詳細は Webサーバー設定を参照してください。

この情報を用いて、サーバーの機能を確認することができ、最終的には対応するパラメーターを適合させます。

注: WEB GET STATISTICSコマンドを使用して、スタティックページに対してキャッシュがどのように使用されているかに関する情報を入手することが可能です。

/4DHTMLSTATS

/4DHTMLSTATS URL は、4DSTATS URLと同じ情報をHTML表形式で返します。その違いはキャッシュされたオブジェクトにHTMLページの情報のみが返され、ピクチャーファイルをカウントしないことです。さらにこのURLは**フィルターされたオブジェクト情報**を返します。

- **フィルターされたオブジェクト**: URLでカウントされないキャッシュ中のオブジェクトの数 (特にピクチャー)。

/4DCACHECLEAR

/4DCACHECLEAR URLは即座にスタティックページとイメージのキャッシュを消去します。そのため、修正されたページを "強制的に" 更新します。

/4DWEBTEST

/4DWEBTEST URLは、Webサーバーの状態を確認するために設計されています。このURLが呼び出されると、4Dは以下のHTTPフィールドを記したテキストファイルのみを返します。

- **Date:** RFC 822フォーマットでの現在の日付
  例えば、"Mon, 16 Jan 2012 13:12:50 GMT"
- **Server:** 4D_version/internal version number番号
  例えば、"4D_v12/12.3"
- **User-Agent:** 名前とバージョン @ IPクライアントアドレス
  例えば、"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:9.0.1) Gecko/20100101 Firefox/9.0.1 @ 127.0.0.1"

接続ログファイル

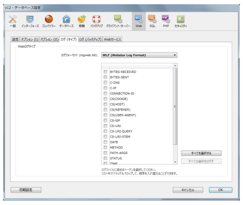
4Dはリクエストのログを提供します。

このファイルには、"logweb.txt" という名前が付けられ自動的に保存されます。

- 4Dのローカルモードと4D Serverでは、データベースストラクチャファイルと同階層のLogsフォルダに配置されます。
- 4Dのリモートモードでは、4Dクライアントデータベースフォルダ (キャッシュフォルダ) のLogsサブフォルダ内に配置されます。

有効化とフォーマット

ログファイルの有効化と内容の設定は、データベース設定のWeb/ログ (タイプ) ページで行います。



v12 - データベース設定

Web / ログ (タイプ)

Webログタイプ

ログフォーマット (logweb.txt): WLF (Webstar Log Format)

- ☐ BYTES-RECEIVED
- ☐ BYTES-SENT
- ☐ C-DNS
- ☐ C-IP
- ☐ CONNECTION-ID
- ☐ CS(COOKIE)
- ☐ CS(HOST)
- ☐ CS(REFERER)
- ☐ CS(USER-AGENT)
- ☐ CS-SIP
- ☐ CS-URI
- ☐ CS-URI-QUERY
- ☐ CS-URI-STEM
- ☐ DATE
- ☐ METHOD
- ☐ PATH-ARGS
- ☐ STATUS
- ☐ TIME

ログファイルに含めるトークンを選択してください。
リスト中でドラッグ＆ドロップして、順序を入れ替えることができます。

注: リクエストのログファイルの有効/無効はSET DATABASE PARAMETER (4D v12) または WEB SET OPTION (4D v13以降) コマンドを使用したプログラミングでも切り替えられます。

ログのフォーマットメニューでは、次のオプションを提供します。

- **No Log File**: このオプションが選択されると、4Dはリクエストのログファイルを作成しません。
- **CLF (Common Log Format)**: このオプションが選択されると、リクエストのログがCLFフォーマットで作成されます。CLFフォーマットでは、以下のように、それぞれのリクエストが行単位でファイル内に表示されます。
  host rfc931 user [DD/MMM/YYYY:HH:MM:SS] "request" state length

  各フィールドはスペースによって区切られ、それぞれの行はCR/LF シーケンス (character 13、character 10) で終ります。
    - host: クライアントのIPアドレス (例: 192.100.100.10)
    - rfc931: 4Dによって作成されない情報。常に - (マイナス記号) です。
    - user: 認証されているユーザ名、あるいは、- (マイナス記号) 。ユーザ名にスペースが含まれると、_ (下線) に置き換わります。
    - DD: 日、MMM: 月を表す3文字の略号 (Jan、Febなど)、YYYY: 年、HH: 時間、MM: 分、SS: 秒
      日付と時間はサーバのローカルタイム。
    - request: クライアントによって送られたリクエスト (例: GET /index.htm HTTP/1.0) 。
    - state: サーバからの返答。
    - length: 返されたデータ (HTTPヘッダー以外) のサイズまたは0

  注: パフォーマンス上の理由により、操作はメモリのバッファに1Kbのパケットで保存され、ディスクに書き込まれます。5秒間リクエストが発生しなくても、操作はディスクに書き込まれます。

  stateとして取り得る値は以下の通り。

  200: OK
  204: No contents
  302: Redirection
  304: Not modified
  400: Incorrect request
  401: Authentication required
  404: Not found
  500: Internal error

  CLFフォーマットはカスタマイズされません。
- **DLF (Combined Log Format)**: このオプションが選択されると、リクエストログがDLFフォーマットで作成されます。DLFフォーマットはCLFフォーマットと類似していて、全く同じス トラクチャを使用します。各リクエストの最後に2つのHTTPフィールド、RefererとUser-agentを追加します。
    - Referer: リクエストされたドキュメントを指しているページのURLを含みます。
    - User-agent: オリジナルリクエストでクライアントのブラウザまたはソフトウェアの名前とバージョンを含みます。

  DLFフォーマットはカスタマイズされません。
- **ELF (Extended Log Format)**: このオプションが選択されると、リクエストログがELFフォーマットで作成されます。ELFフォーマットはHTTPブラウザ界で広く普及しています。そし て、特別なニーズに応える洗練されたログを構築します。この理由により、ELFフォーマットはカスタマイズされます。レコードするフィールドやそのフィー ルドをファイルへ挿入する順番を選択することが可能です。
- **WLF (WebStar Log Format)**: このオプションが選択されると、リクエストログがWLFフォーマットで作成されます。WLFフォーマットは4D WebSTARサーバ用として特別に開発されました。いくつかの追加フィールドを持つので、EFLフォーマットと似ています。EFLフォーマットと同様、カスタマイズされます。

フィールドの設定

ELF (Extended Log Format) フォーマットまたは WLF (WebStar Log Format) を選択すると、"Weg Log Token Selection" エリアで選択されたフォーマットに対して利用可能なフィールドが表示されます。ログに含む各フィールドを選択する必要があります。これを実行するには、矢 印ボタンを使用するか、"Selected Tokens" へ目的のフィールドをドラッグ＆ドロップします。

注: 同じフィールドを2度選択することはできません。

各フォーマットで利用可能なフィールド (アルファベット順) とその内容を以下に示します。

フィールド	ELF	WLF	値
BYTES_RECEIVED		X	サーバが受け取ったバイト数
BYTES_SENT	X	X	サーバがクライアントに送ったバイト数
C_DNS	X	X	DNSのIPアドレス (ELF: C_IP フィールドと同一のフィールド)
C_IP	X	X	クライアントのIPアドレス (例: 192.100.100.10)
CONNECTION_ID		X	接続ID番号
CS(COOKIE)	X	X	HTTPリクエストに格納されているcookiesに関する情報
CS(HOST)	X	X	HTTPリクエストのHostフィールド
CS(REFERER)	X	X	リクエストされたドキュメントを指すページのURL
CS(USER_AGENT)	X	X	ソフトウェアに関する情報とクライアントのオペレーティングシステム
CS_SIP	X	X	サーバのIPアドレス
CS_URI	X	X	リクエストが作成されるURI
CS_URI_QUERY	X	X	リクエストをクエリする引数
CS_URI_STEM	X	X	クエリ引数のないリクエストのパート
DATE	X	X	DD: 日、MMM: 月を表す3文字の略号 (Jan、Febなど)、YYYY: 年
METHOD	X	X	サーバへ送られるリクエスト用のHTTPメソッド
PATH_ARGS		X	CGI引数: "$" の後に続く文字列
STATUS	X	X	サーバの返答
TIME	X	X	HH: 時間、MM: 分、SS: 秒
TRANSFER_TIME	X	X	返答を作成するためにサーバによってリクエストされた時間
USER	X	X	認証された場合はユーザ名。その他の場合は - (マイナス記号) 。 ユーザ名にスペースがある場合、_ (アンダーライン) に置き換えられる。
URL		X	クライアントがリクエストしたURL

注: 日付と時間はGMTで表されます。

周期的なバックアップ

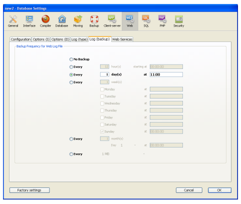
Webのログファイルはかなり膨大になります。そのため、自動のアーカイブメカニズムを構築することが可能です。バックアップはある周期 (時間、日、週、月単位) または、ファイルのサイズに基づいて起動します。設定の期限 (またはファイルサイズ) に近づくと、4Dは自動的にカレントのログファイルを閉じてアーカイブします。そして新たにファイルを作成します。

Webのログファイル用のバックアップが起動すると、ログファイルは "Logweb Archives" という名前のフォルダでアーカイブされます。このフォルダは、logweb.txtファイル (tデータベースストラクチャファイルの隣にあります) と同じレベルで作成されます。

アーカイブされたファイルの名前は、以下の例に基づいて変更されます: "DYYYY_MM_DD_Thh_mm_ss.txt."。例えば、ファイルがアーカイブされた時間がSeptember 4, 2006 at 3:50 p.m. and 7 secondsである場合、"D2006_09_04_T15_50_07.txt." になります。

バックアップパラメタ

リクエストログの自動バックアップは、データベース設定のWeb/ログ (バックアップ) ページで設定します。



new2 - Database Settings

Web / Log (backup)

Backup Frequency for Web Log File

○ No Backup
○ Every 12 hour(s) starting at 00:00:00
● Every 1 day(s) at 11:00
○ Every 1 week(s)
☐ Monday at
☐ Tuesday at
☐ Wednesday at
☐ Thursday at
☐ Friday at
☐ Saturday at
☑ Sunday at 00:00:00
○ Every 1 month(s)
Day 1 at 00:00:00
○ Every 1 MB

最初に、頻度 (日、週などの単位) またはファイルサイズの上限に対応するラジオボタンをクリックして選択します。必要に応じて、バックアップする正確な時間を指定します。

- **バックアップしない**: 設定されたバックアップ機能が無効になっています。
- **X 時間ごと**: 1時間単位でバックアップをプログラムする際、このオプションを使用します。1から24の間の値を入力します。
    - **starting at**: 最初のバックアップの開始時間を設定するために使用します。
- **X 日ごと**: 1日単位でバックアップをプログラムする際、このオプションを使用します。毎日バックアップを実行したい場合、1を入力します。このオプションをチェックすると、バックアップの開始時間を指定しなければなりません。
- **X 週ごと**: 1週間単位でバックアップをプログラムする際、このオプションを使用します。毎週バックアップを実行したい場合、1を入力します。このオプションをチェックすると、バックアップを開始する曜日と時間を指定しなければなりません。必要であれば、複数の曜日を選択することも可能です。例えば、水曜日と金曜日を 選択し、2つのバックアップを設定することができます。
- **X 月ごと**: 1ヶ月単位でバックアップをプログラムする際、このオプションを使用します。毎月バックアップを実行したい場合、1を入力します。このオプションをチェックすると、バックアップを開始する月においての日付と時間を指定しなければなりません。
- **サイズ指定**: カレントのリクエストログのファイルサイズに基づいてバックアップをプログラムする際、このオプションを使用します。ファイルが設定されたサイズに達した際、バックアップが自動的に起動します。サイズ制限は1、10、100または1000MBごとに設定可能です。

Note: バックアップの設定では、バックアップが開始する予定となっているときにWebサーバが起動していない場合、次回の起動において、4Dはバックアップが失敗したと見なし、データベース設定で示されている適切な設定を適用します。

ランタイムエクスプローラの情報

Web サーバーに関連する情報が、ランタイムエクスプローラーにある**ウォッチ**ページ ("Web") に表示されます。

- **Webキャッシュ使用**: Webキャッシュに存在するページ数とその使用率を示します。Webサーバーがアクティブでキャッシュサイズが0より大きい場合のみ、この情報が利用できます。
- **Webサーバー経過時間**: Webサーバーの使用時間を (時間 : 分: 秒フォーマットで) 示します。Webサーバーがアクティブである場合のみ、この情報が利用できます。
- **Webヒット数**: Webサーバーが起動してから受け取ったHTTPリクエストの総数と瞬時のリクエスト数を秒単位で示します (2つのランタイムエクスプローラーの更新の間で測定) 。Webサーバーがアクティブである場合のみ、この情報が利用できます。

注: ランタイムエクスプローラーに関する詳細は、4D Design Referenceマニュアルを参照してください。

SSLプロトコルの使用

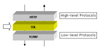
SSL (Secured Socket Layer) プロトコルを通じて、4D Webサーバは保護モードで通信することができます。

SSLプロトコルの定義

SSLプロトコルは2つのアプリケーション、主にWebサーバとブラウザ間でのデータ交換を保護するために設計されています。このプロトコルは幅広く使用されていて、多くのWebブラウザとの互換があります。

ネットワークレベルにおいては、SSLプロトコルはTCP/IPレイヤ (低レベル) とHTTP高レベルプロトコルとの間に挿入されます。SSLは主にHTTPで動作するために指定されます。

SSLを用いたネットワーク設定


HTTP
SSL
TCP/IP
High-level Protocols
Low-level Protocols

注: SSLプロトコルは、標準な4D Serverクライアント/サーバーの接続を保護するためにも使用されます。詳細は、4D Serverリファレンスマニュアルのクライアント/サーバ接続の暗号化とSQLリファレンスマニュアルの4D SQLサーバの設定を参照してください。

SSLプロトコルは、送信者と受信者を認証するために設計され、交換された情報の機密性と整合性を保証します。

- **認証**: 送信者と受信者のIDを確認します。
- **機密性**: 送信データをコード化します。そのため第三者はメッセージを解読することができません。
- **整合性**: 受信データが偶発的にまたは故意に修正されることはありません。

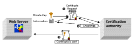
SSLは公開鍵暗号化技術を用います。これは、暗号化と復号化の非対称鍵のペアである公開鍵と秘密鍵に基づいています。
秘密鍵はデータを暗号化するために使用されます。送信者 (Webサイト) は、それを誰にも渡しません。公開鍵は情報を復号化するために使用され、証明書を通して受信者 (Webブラウザー) へ送信されます。インターネットでSSLを使用する際、証明書はVerisign®などの認証機関を通して発行されます。Webサイトは証明書を認証機関から購入します。この証明書はサーバ認証を保証し、保護モードでのデータ交換を許可する公開鍵を格納しています。

注: 暗号化メソッドと公開鍵および秘密鍵に関する詳細は、ENCRYPT BLOBコマンドの記述を参照してください。

証明書の取得方法

4D Webサーバーを保護モードで起動させるには、認証機関の電子証明書が必要です。この証明書にはサイトIDやそのサイトとの通信に使用する公開鍵など様々な情報が格納されます。そのサイトに接続した際に、証明書がWebブラウザーへ送られます。証明書が識別され受け入れられると、保護モードで通信が開始されます。

注: ブラウザーはルート証明書がインストールされた認証機関によって発行された証明書のみを許可します。ルート証明書がインストールされていない場合、通常警告が表示されます。


Web Server
Private Key
Information
❶ Certificate Request
❷ Checkings
❸ Certificate is sent
Certification authority

認証機関は複数の条件によって選択されます。認証機関が一般によく知られていると、証明書は多くのブラウザによって許可されます。ただし支払料金は高くなるかもしれません。

SSL証明書の取得

1. GENERATE ENCRYPTION KEYPAIRコマンドを使用して、秘密鍵を作成します。

警告: セキュリティ上の理由により、秘密鍵は常に機密でなければなりません。実際、秘密鍵は常にWebサーバーマシンと一緒に存在しているべきです。Key.pem ファイルは、データベースストラクチャーフォルダーに保存されていなければなりません。

2. 証明書のリクエストを発行するためにGENERATE CERTIFICATE REQUESTコマンドを使用します。

3. その証明書リクエストを選択された認証機関へ送ります。
証明書リクエストを記入する際、認証機関への問い合わせが必要となる場合があります。認証機関は送信されてきた情報が正確なものかを確認します。その証明書リクエストはPKCSフォーマットを用いてBLOBに作成されます。このフォーマットを使用すると、テキストとしてキーをコピー＆ペーストできます。キーの内容を修正せずに認証機関に提出します。例えばテキストドキュメントに証明書リクエストを含んでいるBLOBを保存します (BLOB TO DOCUMENTコマンドを使用) 。そして、コンテンツを開き、それをコピーして、認証機関へ送信するメールまたはWebフォームにペーストします。

4. 証明書を取得したら、"cert.pem" という名前でテキストファイルを作成し、その証明書の内容をそのファイルへコピーします。
証明書はPKCSフォーマットで、改行コードはLF (Linux形式) でなければなりません。

5. "cert.pem" ファイルをデータベースストラクチャフォルダに保存します。
Webサーバーが保護モードで動作するようになります。証明書は認証機関が設定した期間有効です。

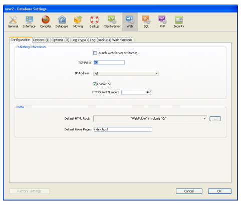
4DへのSSLインストールと起動

4D Webサーバと一緒にSSLプロトコルを使用したい場合、以下の要素が所定の場所にインストールされていなければなりません。

- 4DSLI.DLL (Windows) または 4DSLI.bundle (Mac OS): SSL管理専用のDLL (Secured Layer Interface)
  このファイルはデフォルトでインストールされ、以下の場所に配置されています。
  - Windowsでは、4Dまたは4D Serverアプリケーションの実行ファイルの隣。
  - Mac OSでは、4Dまたは4D Serverパッケージの Native Components サブフォルダ内。
- key.pem (秘密暗号鍵を格納するドキュメント) とcert.pem ("証明書" を格納しているドキュメント):
  - 4Dのローカル モードまたは4D Serverでは、このファイルはデータベースストラクチャファイルと同階層に保存されていなければなりません。
  - 4D のリモートモードでは、このファイルはリモートマシンのデータベースのローカルリソースフォルダに置かれなければなりません。このフォルダの場所に関する 情報は、Get 4D folderコマンドの説明内、4D Client Database Folderの段落を参照してください。これらのファイルは手作業でリモートマシンにコピーしなければならない点に留意してください。

注: また4DSLI.DLLは、暗号化コマンド ENCRYPT BLOB と DECRYPT BLOB を使用するときも必要です。

これらの要素をインストールすることにより、4D Webサーバへ接続するためにSSLの使用を可能にします。ただし、SSL接続が4D Webサーバによって受け入れられるようにするには、SSLを "アクティブ" にしなければなりません。データベース設定の**Web**ページの**設定**タブで、この引数へアクセスできます。


new2 - Database Settings
General
Interface
Compiler
Database
Moving
Backup
Client-server
Web
SQL
PHP
Security
Configuration
Options (I)
Options (II)
Log (type)
Log (backup)
Web Services
Publishing Information
Paths
...
Factory settings
Cancel
OK

☐ Launch Web Server at Startup
TCP Port: 80
IP Address: All
☑ Enable SSL
HTTPS Port Number: 443
Default HTML Root: "WebFolder" in volume "C:"
Default Home Page: index.html

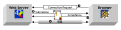
デフォルトで、SSL接続が許可されています。SSLの機能をWebサーバと一緒に使用したくない場合や、安全な接続を許可している他のWebサーバが同じマシンで起動している場合は、このオプションのチェックを外します。

SSLデータ交換専用のTCPポートはデフォルトで443です。例えば、Webサーバのセキュリティを強化する場合など、**HTTPSポート番号**エリアでこのポート番号を修正することができます (このポートに関する詳細は Webサーバー設定 を参照してください) 。データベース設定のページで定義されたTCPポートは、標準モードのWebサーバ接続の際に使用されます。

Note: SSLモードでサーバが起動していても、していなくても、4D Web サーバ管理 (パスワード、タイムアウト、キャッシュサイズなど) 用に定義された別の設定が適用されます。

SSLでブラウザの接続

保護モードで実行されるWeb接続に対しては、ブラウザによって送られるURLは、"https" ("http" の代わりとして) で始まる必要があります。この場合、ブラウザに警告ダイアログが現れます。ユーザーが**OK**をクリックすると、Webサーバーはブラウザに証明書を送ります。


Web Server
❶ Connection Request
❷ Submission
❸ Acceptance
❹ SSL
Browser

接続するために使用する暗号化アルゴリズムは、ブラウザとWebサーバによって決定されます。サーバはいくつかの相称的な暗号化アルゴリズム (RC2、RC4、DESなど)を提供します。一番効果のある共通なアルゴリズムが使用されます。

警告: 許可された暗号化のレベルは、使用する国の現行の法律によって異なります。4D Web サーバによって提供された暗号化のレベルは、使用する暗号化システムライブラリのバージョンに応じます。

接続モードの管理

4D WebサーバーとのSSLの使用は、特別なシステム環境の設定を必要としません。ただしSSL Webサーバーは非保護モードにおいても動作するということを覚えておいてください。例えば、ブラウザーが要求するのであれば、接続モードを他のモードへ切り 換えることが可能です (例としてブラウザーURLエリアで、ユーザーは "HTTPS" を "HTTP" と置き換えることができます) 。デベロッパーは、非保護モードで作成されたリクエストを禁じたり、リダイレクトすることができます。WEB Is secured connectionコマンドを使用すると、現在の接続モードを得ることが可能です。

XMLとWMLサポート

WML

4D Webサーバは、WML (Wireless Markup Language) テクノロジーをサポートしています。このテクノロジーは、携帯電話やPDAで4Dのデータベース上のデータを読んだり、入力したりすることを可能にします。

注: WAP (Wireless Application Protocol) に関連するWMLランゲージは複数の企業によって開発されています。WAPテクノロジーは、一連のネットワークコミュニケーションツールを提供しているため、携帯電話やPDAユーザがWebページ上で公開されているテキストを視覚することができます。
WMLテクノロジーは無料で自由に入手できます。WMLに関する詳細は、Phone.comのWebサイトを参照してください。

WMLページを通してデータを入力したり読んだりするには***4DTEXT***や***4DSCRIPT***等のタグを用います。
以下は、4D WebサーバによってサポートされているWMLドキュメントのリストです。

拡張子	MIMEタイプ	説明
.wml	text/vnd.wap.wml	WML ページ (常に4Dによってサポートされています*)
.wmls	text/vnd.wap.wmlscript	WML スクリプト (クライアント側で)
.wmlc	application/vnd.wap.wmlc	WML バイナリページ
.wmlsc	application/vnd.wap.wmlscript	WML バイナリページ
.wbmp	picture/vnd.wap.wbmp	携帯電話用のビットマップイメージ (常にサポートサポートされているわけではありません)

****4DTEXT***と***4DSCRIPT***等のタグによるダイナミックデータの挿入を可能にします。

XML

4D Web サーバは.xml、.xlsおよび.dtdドキュメントをサポートします。これらのドキュメントはMIMEタイプ "text/xml" と "text/xsl" で送られます。

4Dはドキュメントの内容を解析し、4DTEXTや4DSCRIPTタイプのタグ (もしあれば) を処理してダイナミックXMLを作成します。

WEB CLOSE SESSION

WEB CLOSE SESSION (sessionID)

引数	型		説明
sessionID	テキスト	→	セッションUUID

説明

WEB CLOSE SESSIONコマンドはsessionID引数で指定された既存のセッションを破棄します。指定されたセッションが存在しない場合、コマンドはなにも行いません。

Webプロセスや他のプロセスからこのコマンドが呼び出されると:

- ブラウザーに送信されるcookieの有効期限が0に設定されます。
- 開発者がセッション情報を保存できるようにするためにOn Web Session Suspend データベースメソッドが呼び出されます。
- カレントセレクションや変数などのプロセスオブジェクトが消去され、レコードのロックが解除されます。

このコマンド実行後、Webクライアントが当該cookieを使用して4D Webサーバーにアクセスすると、新しいセッションが開始され新しいcookieがクライアントに送信されます。

WEB GET BODY PART

WEB GET BODY PART (part ; contents ; name ; mimeType ; fileName)

引数	型		説明
part	倍長整数	→	パート番号
contents	BLOB, テキスト	←	パートを受け取る変数
name	テキスト	←	input要素のname属性値
mimeType	テキスト	←	ファイルのMIMEタイプ
fileName	テキスト	←	送信されたファイルの名前

説明

WEB GET BODY PARTコマンドはWebプロセスのコンテキストで使用され、マルチパートリクエストのボディ部を解析します。

part 引数には解析対象のパート番号を渡します。総パート数はWEB Get body part countコマンドで取得できます。

contents 引数にはパートのコンテンツが返されます。取得するパートがファイルの場合、BLOB型の引数を渡さなければなりません。Webフォームから送信されるテキストデータの場合、テキスト型の引数を渡すことができます。

name 引数にはフォームのinput 要素のname属性値が返されます。

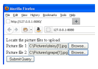
mimeType と fileName 引数には、送信されたファイルのMIMEタイプと名前が返されます。fileName はフォーム要素 **<input type="file">** を使用してファイルが送信された場合のみ値が返されます。
mimeType と fileName はオプションですが、使用する場合はペアで渡さなければなりません。

注: マルチパートリクエストのコンテキストでは、WEB GET VARIABLESコマンドで取得できる名前配列にはフォームのすべてのパートが含まれます。順番は**WEB GET BODY PART**コマンドと同じです。フォーム中のパートの位置を取得するためにこのコマンドを使用できます。

警告: mimeType と fileName 引数に返される情報は、Webサーバーに送信されたHTTPリクエストに基づくもので、ファイルの内容は検証されません。それらの情報が正当であるかどうかの検証は開発者が行わなければなりません。

例題

この例題ではWebフォームからHTTPサーバーに画像を何枚かアップロードし、それらを返されたWebページ上に表示します。Webフォームは以下のように表示されます:


Locate the picture files to upload:
Picture file 1: C:\Pictures\daisy[1].jpg
Picture file 2: C:\Pictures\grape[1].jpg

HTMLソースは以下の通りです:

```
<body>
        <form enctype="multipart/form-data" action="/4DACTION/GetFile/" method="post">
            アップロードするピクチャーを選択: <br>
            ピクチャーファイル1: <input name="file1" type="file"><br>
            ピクチャーファイル2: <input name="file2" type="file"><br>
            <input type="submit">
        </form>
        <hr/>
    <!--4DSCRIPT/galleryInit-->
    <!--4DLOOP aFileNames-->
        <img src="/photos/<!--4DTEXT aFileNames{aFileNames}-->"/>
    <!--4DENDLOOP-->
</body>
```

2つの4Dメソッドがページから呼ばれています:

- ページを送信する際、4DSCRIPタグで呼び出される galleryInit 。このメソッドは指定されたフォルダー内に存在するピクチャー名の配列を作成します。
- ブラウザーからのリクエストを処理する GetFile メソッド。

galleryInitのコードは以下の通りです:

```
 C_TEXT($vDestinationFolder)
 ARRAY TEXT(aFileNames;0)
 C_LONGINT($i)
 $vDestinationFolder:=Get 4D folder(HTML Root Folder)+"photos"+Folder separator //"WebFolder/photos" フォルダー
 DOCUMENT LIST($vDestinationFolder;aFileNames)
```

GetFileのコードは以下の通りです:

```
 C_TEXT($vPartName;$vPartMimeType;$vPartFileName;$vDestinationFolder)
 C_BLOB($vPartContentBlob)
 C_LONGINT($i)
 $vDestinationFolder:=Get 4D folder(HTML Root Folder)+"photos"+Folder separator
 For($i;1;WEB Get body part count) // パートの数だけ繰り返す
    WEB GET BODY PART($i;$vPartContentBlob;$vPartName;$vPartMimeType;$vPartFileName)
    If($vPartFileName#"")
  // ここでファイルの内容を検証すべきです
       BLOB TO DOCUMENT($vDestinationFolder+$vPartFileName;$vPartContentBlob)
    End if
 End for
 WEB SEND HTTP REDIRECT("/")
```

WEB Get body part count

WEB Get body part count -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	ボディ中のパート数

説明

WEB Get body part countコマンドは受信したボディに含まれるパートの数を返します。

例題

WEB GET BODY PART コマンドの例題を参照してください。

WEB Get Current Session ID

WEB Get Current Session ID -> 戻り値

引数	型		説明
戻り値	テキスト	↩	セッションUUID

説明

WEB Get Current Session IDコマンドはカレントのWebリクエストのセッションIDを返します。このIDは4Dが自動で生成します。

このコマンドがWebセッション管理のコンテキストの外で呼び出されると、コマンドは空の文字列を返します。

WEB GET HTTP BODY

WEB GET HTTP BODY (body)

引数	型		説明
body	BLOB, テキスト	←	HTTPリクエストのボディ

説明

WEB GET HTTP BODYコマンドは、処理中のHTTPリクエストのボディを返します。HTTPボディは、処理や解析されることなく、そのままの状態で返されます。

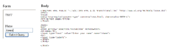
このコマンドはWebデータベースメソッド (On Web Authenticationデータベースメソッド、On Web Connectionデータベースメソッド) 、またはWebリクエスト処理メソッドで呼び出します。

引数 body にはBLOBまたはテキストタイプの変数やフィールドを渡すことができます。一般的に、テキスト型引数の利用が推奨されます (body引数は2GBを上限としたテキストを受け取ることができます)。

例えばこのコマンドを使用して、リクエストのボディ内を検索することができます。また上級ユーザーは、4DのデータベースでWebDAVサーバーを設定することもできます。

例題

4D Webサーバーに簡単なリクエストを送り、HTTPボディの内容をデバッガーで表示します。
以下は4D Webサーバーに送られたフォームと対応するHTMLコードです。



次はTest4Dv11メソッドです。

```
C_TEXT($requestText)

WEB GET HTTP BODY($requestText)
WEB SEND FILE("page.html")
```

注: このメソッドのプロパティに"4DHTMLタグおよびURL (4DACTION) で利用可能"を指定します。

フォームがWebサーバーに提出されると、変数$requestTextはHTTPリクエストボディのテキストを受け取ります。

WEB GET HTTP HEADER

WEB GET HTTP HEADER (header|fieldArray {; valueArray})

引数	型		説明
header|fieldArray	テキスト, テキスト配列	←	リクエストHTTPヘッダまたはHTTPヘッダフィールド
valueArray	テキスト配列	←	HTTPヘッダフィールドの内容

説明

WEB GET HTTP HEADERコマンドは、現在処理されているリクエストのHTTPヘッダーを含む2つの配列、または文字列のいずれかを返します。

このコマンドはWebプロセスで実行されるすべてのメソッド ('/4DACTION'...によって呼び出されるメソッド、On Web Authenticationデータベースメソッドまたは<u>On Web Connectionデータベースメソッド</u>) 内から呼び出されます。

- **1番目のシンタックス: WEB GET HTTP HEADER (header)**
  このシンタックスを使用すると、次の結果 (例) が変数 <u>header</u> に返されます。
  "GET /page.html HTTP/1.0[CRLF]User-Agent: browser[CRLF]Cookie: C=HELLO"

Windows と Mac OS上で、各ヘッダーフィールドはCR+LF (キャリッジリターン+ラインフィード) シーケンスによって区切られています。

- **2番目のシンタックス: WEB GET HTTP HEADER (fieldArray; valueArray)**

  このシンタックスを使用すると、次の結果がfieldArrayと <u>valueArray</u> に返されます。

fieldArray{1} = "X-METHOD"	valueArray{1} = "GET" *
fieldArray{2} = "X-URL"	valueArray{2} = "/page.html" *
fieldArray{3} = "X-VERSION"	valueArray{3} = "HTTP/1.0" *
fieldArray{4} = "User-Agent"	valueArray{4} = "browser"
fieldArray{5} = "Cookie"	valueArray{5} = "C=HELLO"

* 最初の3つのアイテムはHTTPフィールドではありません。リクエストの最初の行の一部です。

HTTP標準に準拠するには、フィールド名を常に英語で記述します。

リクエストで使用可能ないくつかのHTTPフィールドを以下のリストに示します。

- **Accept**: ブラウザーが許可するコンテンツ
- **Accept-Language**: ブラウザーが使用するランゲージ。ブラウザー上で定義されたランゲージでWebページを選択できます。
- **Cookie**: cookiesリスト
- **From**: ブラウザーユーザーemailアドレス
- **Host**: サーバー名またはアドレス (例えば、URLがhttp://mywebserver/mypage.htmlとすると、**Host**はmywebserverとなります) 。同じIPアドレスを示す複数の名前を管理することができます (バーチャルホスティング) 。
- **Referer**: そのリクエストを発行した元のURL (例えば、http://mywebserver/mypage1.html)。つまりブラウザーの**戻る**ボタンをクリックした際に表示されるページ。
- **User-Agent**: ブラウザーまたはプロキシの名前とバージョン。

例題

次のメソッドを使用して、任意のHTTPリクエストヘッダーのコンテンツを取得できます。

```
  // GetHTTPFieldプロジェクトメソッド
  // GetHTTPField (Text) -> Text
  // GetHTTPField (HTTP header name) -> HTTPヘッダーコンテンツ

 C_TEXT($0;$1)
 C_LONGINT($vlItem)
 ARRAY TEXT($names;0)
 ARRAY TEXT($values;0)
 $0:=""
 WEB GET HTTP HEADER($names;$values)
 $vlItem:=Find in array($names;$1)
 If($vlItem>0)
    $0:=$values{$vlItem}
 End if
```

このプロジェクトメソッドを記述したら、以下のように呼び出しを行います。

```
  // Cookieヘッダーコンテンツ
 $cookie:=GetHTTPField("Cookie")
```

ブラウザー上で設定されたランゲージに応じて、異なるページを送ることができます (例えば、On Web Connection データベースメソッド において) 。

```
 $language:=GetHTTPField("Accept-Language")
 Case of
    :($language="@ja@") // 日本語 (ISO 639リストを参照)
       WEB SEND FILE("index_ja.html")
    :($language="@sp@") // スペイン語 (ISO 639リストを参照)
       WEB SEND FILE("index_es.html")
    Else
       WEB SEND FILE("index.html")
 End case
```

注: Webブラウザ上で複数のランゲージをデフォルトで定義することができます。ランゲージは"Accept-Language" フィールドでリストにされ、";" で区切られて示されています。文字列内でのランゲージの位置に応じて、その優先順位が定義されます。そのため、文字列内でのランゲージの位置をテストすることをお勧めします。

以下は仮想ホストの例です (例えば<u>On Web Connectionデータベースメソッド</u>において) 。次の名前"home_site.com"、"home_site1.com" と "home_site2.com" は同じIPアドレス、例えば192.1.2.3を指している。

```
 $host:=GetHTTPField("Host")
 Case of
    :($host="www.site1.com")
       WEB SEND FILE("home_site1.com")
    :($host="www.site2.com")
       WEB SEND FILE("home_site2.com")
    Else
       WEB SEND FILE("home_site.com")
 End case
```

WEB GET OPTION

WEB GET OPTION (selector ; value)

引数	型		説明
selector	倍長整数	→	取得するオプションのコード
value	倍長整数, テキスト	←	オプションの値

説明

WEB GET OPTIONコマンドは4D Webサーバー処理に関するオプションの現在の設定値を取得するために使用します。

selector 引数には取得するWebオプションを指定する値を渡します。Web Serverテーマの以下の定数を使用できます:

定数	型	値	コメント
Web Character set	倍長整数	17	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: データベースに接続するブラウザーとの通信に (ローカルモード4Dと4D Serverの) Webサーバーが使用する文字セット。デフォルト値はOSの言語に依存します。この引数はデータベース設定でも設定できます。 **値**: 取りうる値は、文字セットに関連するデータベースの動作モードによります。 • Unicode モード: アプリケーションがUnicodeモードで動作している場合、この引数に渡す値は文字セット識別子です。(*MIBEnum*倍長整数または文字列。以下のアドレスを参照: http://www.iana.org/assignments/character-sets)。以下は4D Webサーバがサポートする文字セットに対応する識別子のリストです: 4=ISO-8859-1 12=ISO-8859-9 13=ISO-8859-10 17=Shift_JIS 2024=Windows-31J 2026=Big5 38=euc-kr 106=UTF-8 2250=Windows-1250 2251=Windows-1251 2253=Windows-1253 2255=Windows-1255 2256=Windows-1256 **注**: IANAに定義されていない特別な文字セット (1258=x-mac-japanese) を使用することができます。これはWindows上ではコードページ10001に、Mac上ではkTextEncodingMacJapaneseにマップされています。 • *ASCII 互換モード*: 0: Western European 1: Japanese 2: Chinese 3: Korean 4: User-defined 5: Reserved 6: Central European 7: Cyrillic 8: Arabic 9: Greek 10: Hebrew 11: Turkish 12: Baltic
Web HTTP Compression Level	倍長整数	50	**スコープ**: *ローカルWebサーバー* **2セッション間で設定を保持**: No **説明**: 4D HTTPサーバーで使用されるすべての圧縮されたHTTP通信 (クライアントのリクエスト、サーバーのレスポンス、WebおよびWebサービス) の圧縮レベル。このセレクターを使用すれば圧縮率を犠牲にして実行速度を速めるか、速度を優先して圧縮率を高めるかを選択できます。値の選択は交換するデータのサイズやタイプに基づきます。value引数に1から9までの値を渡します。1は速度優先、9は圧縮率優先です。また-1を渡して圧縮速度と圧縮率の妥協点を指定できます。デフォルトの圧縮レベルは1 (速度優先) です。 **とりうる値**: 1 から 9 (1 = 速度優先, 9 = 圧縮優先) または -1 = 最適
Web HTTP Compression Threshold	倍長整数	51	**スコープ**: ローカルHTTPサーバー **2セッション間で設定を保持**: No **説明**: 最適化モードの内部的な4D Webサービス通信フレームワークにおいて、圧縮を行わないリクエストサイズの敷居値を設定できます。この設定は、小さなデータ交換時に圧縮を行うことによる、マシンの時間の浪費を避けるために有効です。 **とりうる値**: 任意の倍長整数値。valueにはバイト単位でサイズを渡します。デフォルトで、圧縮の敷居値は1024バイトに設定されています。
Web HTTPS Port ID	倍長整数	39	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: ローカルモード4Dおよび4D ServerのWebサーバがSSLによるセキュアな接続（HTTPSプロトコル）で使用するTCPポート番号。HTTPS ポート番号はデータベース設定の"Web/設定"ページで指定できます。デフォルト値は443 (標準ポート番号) です。value 引数にTCP Port Numbersテーマの定数を渡すこともできます。 **とりうる値**: 0~65535
Web Inactive process timeout	倍長整数	78	**スコープ**: ローカルWebサーバー **2セッション間で値を保持**: No. しかしHTTPサーバーを再起動しただけの場合は保持されます。 **説明**: セッション管理のために使用されるプロセスのタイムアウトを設定します。タイムアウト後、プロセスは終了されます。 **取りうる値**: 倍長整数 (分) **デフォルト値**: 480分 (= 8時間) (0を渡すとデフォルト値に設定されます)
Web Inactive session timeout	倍長整数	72	**スコープ**: ローカルWebサーバー **2セッション間で値を保持**: No. しかしHTTPサーバーを再起動しただけの場合は保持されます。 **説明**: セッション管理のために使用されるcookieのタイムアウトを設定します。 **取りうる値**: 倍長整数 (分) **デフォルト値**: 480分 (= 8時間) (0を渡すとデフォルト値に設定されます)

Web IP Address to listen	倍長整数	16	**スコープ**: 4D ローカル, 4D Server
			2セッション間で設定を保持: Yes
			説明: ローカルモード4Dおよび4D ServerのWebサーバがHTTPリクエストを受信するIPアドレス。デフォルトで、特定のアドレスは定義されていません（value=0）。この引数はデータベース設定で設定できます。
			Web IP Address to listenセレクターは、コンパイルして4D Volume Desktopを組み込んだ4D Webサーバで役立ちます (この場合デザインモードへのアクセス手段がありません)。
			value引数には16進数のIPアドレスを渡します。つまり、"a.b.c.d"のようなIPアドレスを指定するには、以下のようなコードを作成します:
			`C_LONGINT($addr)`
			`$addr:=($a<<24)｜($b<<16)｜($c<<8)｜$d`
			`SET DATABASE PARAMETER(Web IP Address to listen;$addr)`

Web Keep session	倍長整数	70	**スコープ**: ローカルWebサーバー **2セッション間で設定を保持**: No. しかしHTTPサーバー再起動後も設定は有効。 **説明**: 4Dによる自動セッション管理モード (Webセッション管理) の有効/無効を設定する。 **取りうる値**: 1 (有効) / 0 (無効) **デフォルト値**: v13で作成されたデータベースでは1、変換されたデータベースでは0。このモードはリモートモードで一時的なコンテキストの再利用メカニズムも有効にする点に留意してください。このメカニズムに関する詳細はWebサーバー設定を参照してください。
Web Log Recording	倍長整数	29	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: ローカルモード4Dまたは4D ServerのWebサーバーが受け取るWebリクエストの記録を開始または停止します。デフォルト値は0(リクエストを記録しない)です。 Web リクエストのログは"logweb.txt"という名前のテキストファイルに保存されます。このファイルは自動でストラクチャファイルと同階層のLogsフォ ルダー内に作成されます。このファイルのフォーマットは、渡した値により決定されます。Webログファイルフォーマットに関する詳細はWebサイトに関する情報を参照してください。 このファイルはデータベース設定の"Web/ログ"ページからも有効にできます。 **とりうる値**: 0 = 記録しない (デフォルト), 1 = CLFフォーマットで記録, 2 = DLFフォーマットで記録, 3 = ELFフォーマットで記録, 4 = WLFフォーマットで記録 **警告**: フォーマット3および4はカスタムフォーマットであり、記録する内容を事前にデータベース設定で定義しなければなりません。事前定義せずにこれらのフォーマットを使用した場合、ログファイルは作成されません。
Web Max Concurrent Processes	倍長整数	18	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: ローカルモード4Dならびに4D ServerのWebサーバーでサポートされる、任意のタイプの同時Webプロセス上限数を厳密に設定。この上限数（マイナス1）に達した場合、4Dはそれ以上プロセスを作成しなくなり、HTTPステータス503（Service Unavailable）をすべての新しいリクエストに返します。 このパラメーターにより、同時に行われる非常に膨大な数のリクエストやコンテキスト作成に関する過大な要求の結果として、4D Webサーバが飽和状態になることを防ぐことができます。このパラメーターはデータベース設定でも設定できます。 理論上、Webプロセスの最大数は次の計算式の結果になります: 使用可能メモリ/Webプロセスのスタックサイズ。別の解決策は、ランタイムエクスプローラに表示されるWebプロセス情報を監視する方法です。つまり現在のWebプロセス数およびWebサーバの開始以降に達した最大数を監視します。 **値**: 10から32 000までの任意の数。デフォルト値は100。
Web Max sessions	倍長整数	71	**スコープ**: ローカルWebサーバー **2セッション間で設定を保持**: No. しかしHTTPサーバー再起動後も設定は有効。 **説明**: 4Dの自動セッション管理下のセッション上限数。設定した上限に達すると、もっとも古いセッションが閉じられます。 **取りうる値**: 倍長整数値 **デフォルト値**: 100 (0を渡すとデフォルト値が設定されます)
Web Maximum requests size	倍長整数	27	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: Webサーバに処理を許可する受信HTTPリクエスト (POST) の最大サイズ (バイト単位)。デフォルト値は2,000,000 (2MBより少し少ない値) です。最大値 (2,147,483,648) を渡すと、実際上は制限がなくなります。 この制限は、受信するリクエストが大きすぎるためにWebサービスが飽和してしまうことを回避するために使用します。リクエストがこの制限に達すると、4D Webサービスはリクエストを拒否します。 **とりうる値**: 500,000～2,147,483,648.
Web Session cookie name	倍長整数	73	スコープ: ローカルWebサーバー 2セッション間で設定を保持: No. しかしHTTPサーバー再起動後も設定は保持される。 説明: 4Dの自動セッション管理機能で使用されるcookieのname属性値。 取りうる値: テキスト デフォルト値: "4DSID" (空文字を渡すとデフォルト値が設定されます。)

注 (v13.2): selector 引数に使用できるコードが追加されました。将来のバージョンの4Dではこれらのコードのための定数が追加される予定です。v13.2では数値を指定することで使用できます。詳細はWEB SET OPTIONを参照してください。

WEB GET SESSION EXPIRATION

WEB GET SESSION EXPIRATION (sessionID ; expDate ; expTime)

引数	型		説明
sessionID	テキスト	⇨	セッションUUID
expDate	日付	⇦	cookie有効期限日
expTime	時間	⇦	cookie有効期限時刻

説明

WEB GET SESSION EXPIRATIONコマンドはsessionIDに渡されたUUIDのセッションのcookieの有効期限に関する情報を返します。

expDate 引数はcookieの有効期限日を、expTime 引数はcookieの有効期限時刻を受け取ります。

注: Webレスポンスがクライアントに送信されるたびに、cookieの有効期限はリクエストが行われた時刻+Web Inactive session timeout (デフォルトで8時間) に設定されます。例えばデフォルト値の状態で:
最初のリクエスト: 月曜日の1:00
  -> 有効期限は月曜日の09:00
二番目のリクエスト: 月曜日の1:10
  -> 有効期限は月曜日の09:10
三番目のリクエスト: 火曜日の4:00 (cookieの有効期限が過ぎている)
  -> 新しいcookie値が生成され、有効期限は火曜日の12:00

WEB GET STATISTICS

WEB GET STATISTICS (pages ; hits ; usage)

引数	型		説明
pages	テキスト配列	←	最も閲覧されるページの名前
hits	倍長整数配列	←	各ページのヒット数
usage	倍長整数	←	キャッシュの使用率

説明

WEB GET STATISTICSコマンドを使用すると、Webサーバのキャッシュに読み込まれた最も閲覧されたページに関する情報を取得 することができます。そのため、これらの統計は、静止画、GIFピクチャ、JPEGピクチャ (100KB未満) とスタイルシート (.css) のみに適用されます。

Note: 4D Webサーバのキャッシュの設定に関する詳細はQR DELETE COLUMNを参照してください。

コマンドは最も閲覧されたページの名前をテキスト配列pagesに代入します。倍長整数配列 hits は各ページの "ヒット" 数を受け取ります。引数 usage は各ページごとのWebキャッシュの使用率を受け取ります。

例題

Webキャッシュの統計を表示するセミダイナミックなページを生成します。これを行うために、"stats.shtm"という名前のスタティックHTMLページ (4Dは自動で.shtm拡張子のファイルを解析対象とします) 中に*<!—#4DSCRIPT/STATS—>*タグと、*vPages*と*vUsage*変数への参照を記述します。例えば

```
<html>
<head><title>4D Web統計</title></head>
<!--#4DSCRIPT/STATS-->
<body>
<strong>使用されているキャッシュ (%): </strong>
<!--#4DTEXT vUsage-->
<hr>
<strong>最も参照されているページ: </strong>
<!--#4DHTML vPages-->
</body>
</html>
```

STATSプロジェクトメソッドには以下のコードを書きます:

```
C_TEXT($1)
C_TEXT(vPages)
ARRAY TEXT(pages;0)
ARRAY LONGINT(hits;0)
C_LONGINT(vUsage)

WEB CACHE STATISTICS(pages;hits;vUsage)
For($i;1;Size of array(pages))
  // キャッシュ中に現れるページごとに
    vPages:=vPages+pages{$i}+"   "+String(hits{$i})+"<br>"
  // ページの名称とHTMLコードを挿入する
End for
```

URLリンクまたはWEB SEND FILEコマンドを使用して"stats.shtm"ページを参照できます。

WEB GET VARIABLES

WEB GET VARIABLES (nameArray ; valueArray)

引数	型	説明
nameArray	テキスト配列	← Webフォームの変数の名前
valueArray	テキスト配列	← Webフォームの変数の値

説明

WEB GET VARIABLESコマンドは、サブミットされたWebフォームにある変数の名前と値をテキスト配列 nameArrayとvalueArray に代入します。

このコマンドはHTMLページ、テキストエリア、ボタン、チェックボックス、ラジオボタン、ポップアップメニュー、選択リストにあるすべての変数の値を取得します。

注: チェックボックスに関しては、チェックボックスが実際にチェックされた場合に限り、その変数の名前と値が返されます。

URLのタイプやWebサーバーに送られたフォーム (POSTまたはGETメソッド) に関係なく有効です。

On Web Connectionデータベースメソッドや、フォームをサブミットすることによって呼び出される他の4Dメソッドにおいて、必要に応じこのコマンドを呼び出します。

Webフォームと関連するアクションについて

各フォームには、名前が付けられたデータを入力するエリアがあります (テキストエリア、ボタン、チェックボックス) 。
フォームがサブミットされると (リクエストがWebサーバへ送られます) 、リクエストはデータ入力エリアのリストとそれに関連する値を格納します。
2つのメソッドでフォームをサブミットできます (両方とも4Dで使用できます) 。

- POST: 通常Webサーバやデータベースにデータを追加する際に使用します。
- GET: 通常Webサーバやデータベースから抽出されたデータをリクエストする際に使用します。

例題

フォームには、値 "ROBERT" と "DALLAS" を持つ vNameフィールドとvCityフィールドがそれぞれあります。そのフォームに関連するアクションは、"/4DACTION/WEBFORM" です。

- フォームメソッドがPOST (頻繁に使用されます) である場合、入力されたデータはURL上で表示されません。
- フォームメソッドがGETである場合、入力されたデータはURLで表示されます。(http://127.0.0.1/4DACTION/WEBFORM?vNAME=ROBERT&vCITY=DALLAS).

WEBFORMメソッド以下のように記述します。

```
ARRAY TEXT($anames;0)
ARRAY TEXT($avalues;0)
WEB GET VARIABLES($anames;$avalues)
```

結果は以下のようになります。

```
$anames{1}="vNAME"
$anames{2}="vCITY"
$avalues{1}="ROBERT"
$avalues{2}="DALLAS"
```

変数vNAMEは、ROBERTを格納し、変数vCITYは、DALLASを格納します。.

WEB Is secured connection

WEB Is secured connection -> 戻り値

引数	型		説明
戻り値	ブール	↩	True = Web接続が保護されている場合 False = Web接続が保護されていない場合

説明

WEB Is secured connectionコマンドは、4DのWebサーバ接続が (リクエストが"http:"の代わりに"https:"で始まる) SSLを通して保護モードで実行されたかを示すブール値を返します 。

- SSLを経由して接続された場合、関数はTrueを返します。
- 非保護モードで接続された場合、関数はFalseを返します。

Note: SSLプロトコルに関する詳細はSSLプロトコルの使用を参照してください。

このコマンドを使用して、非保護モードで実行された接続を拒否することも可能です。

WEB SEND BLOB

WEB SEND BLOB (BLOB ; type)

引数	型	説明
BLOB	BLOB	→ブラウザへ送るBLOB
type	文字	→BLOBのデータタイプ

説明

WEB SEND BLOBコマンドを使用して blob をブラウザへ送ります。

BLOBに含まれるデータのタイプは type によって示されます。この引数は以下の定数のいずれかになります:

- Type = **空の文字列** (""): この場合、BLOBにそれ以上の情報を供給する必要はありません。ブラウザがBLOBの内容を解釈しようとします。
- Type = **ファイル拡張子** (例: ".HTM"、".GIF"、".JPEG" など): この場合、BLOBに含まれるデータのMIMEタイプを拡張子を使って指定します。そして、BLOBはその拡張子に応じて解釈されます。ただし、ブラウザ が正確に解釈できるように、拡張子は標準なものでなくてはなりません。
- type = **Mime/Type** (例: "text/html"、"image/tiff" など):この場合、BLOBに含まれるデータのMIMEタイプを直接指定します。このソリューションはより多くの自由度を提供します。標準タイプに加え、イントラネット経由で固有のドキュメントを送るためにカスタムMIMEタイプを渡すこともできます。これを実行するには、ブラウザを 設定するだけです。このブラウザが、送られたタイプを認識したり、適切なアプリケーションを開いたりします。 type isに渡す値は、この場合、"application/x-[TypeName]"となります。クライアントワークステーションのブラウザでは、このタイプを参照し、それを "アプリケーション起動" 動作に関連付けます。そのため**WEB SEND BLOB**コマンド を使用すると、すべてのタイプのドキュメントを送ることが可能になり、イントラネットクライアントは関連するアプリケーションを自動的に開くことができるようになります。

  注: MIME型についての詳細な情報に関しては、こちらを参照して下さい: http://www.iana.org/assignments/media-types

最も一般的なMIMEタイプのリストです:

拡張子	Mime/Type
.htm	text/html
.html	text/html
.shtml	text/html
.shtm	text/html
.css	text/css
.pdf	application/pdf
.rtf	application/rtf
.ps	application/postscript
.eps	application/postscript
.hqx	application/mac-binhex40
.js	application/javascript
.json	application/json
.txt	text/plain
.text	text/plain
.gif	image/gif
.jpg	image/jpeg
.jpeg	image/jpeg
.jpe	image/jpeg
.jfif	image/jpeg
.pic	image/pict
.pict	image/pict
.tif	image/tiff
.tiff	image/tiff
.mpeg	video/mpeg
.mpg	video/mpeg
.mov	video/quicktime
.moov	video/quicktime
.aif	audio/aiff
.aiff	audio/aiff
.wav	audio/wav
.ram	audio/x-pn-realaudio
.sit	application/x-stuffit
.bin	application/x-stuffit
.xml	application/xml
.z	application/x-zip
.zip	application/x-zip
.gz	application/x-gzip
.tar	application/x-tar

注: 4D HTTPサーバーでサポートされているMIMEタイプは、4Dアプリケーションの以下のフォルダの中にある、"MimeTypes.xml" の中に保存されています: *[Contents]\Native components\HTTPServer.bundle\Contents\Resources.*

ページ内の4D 変数や 4DSCRIPT 型タグへの参照は常に解析されます。

例題

PICTURE TO GIFルーチンの例を参照してください。

WEB SEND FILE

WEB SEND FILE (htmlFile)

引数	型		説明
htmlFile	文字	→	HTMLファイルへのHTMLパス名 または、WEB SEND FILEを終了させる場合空の文字列

説明

WEB SEND FILEコマンドはhtmlFileに渡すパス名を持つHTMLページやWebファイルをWebブラウザへ送ります。

デフォルトで、4Dはドキュメントをデータベース設定で定義されているHTMLルートフォルダ内で検索します。

コマンドは (ディレクトリやフォルダー名がスラッシュで区切られる) Posixシンタックスまたはシステムシンタックスを受け入れます。
無効なパス名を指定すると、4DはOSのファイル管理に関連するエラーを生成します。このエラーをON ERR CALLでインストールするエラー処理メソッドでとらえることができます。メソッドがダイアログボックスや警告を表示すると、それはWebブラウザに表示されます。

WEB SEND FILEコマンドが実行されると、OKシステム変数が更新されます。送るファイルが存在していて、タイムアウトになっていなければ、OKシステム変数は1になります。その他の場合は0になります。

注: WEB SEND FILEコマンドをWebプロセスでないプロセスから呼び出した場合、コマンドは何も行わず、エラーも返しません。呼び出しは無効となります。

ドキュメントのタイプが対応する場合、送信前に*4DSCRIPT*等のタグが解析されます。

例題

データベースのHTMLルートフォルダはストラクチャーと同階層の**WebFolder**に設定されていて、以下のディレクトリにファイルが置かれています。

```
../WebFolder/HTM\MyPage.HTM
```

Webページ "MyPage.HTM"を送信するには以下のようにします。

```
WEB SEND FILE("HTM/MyPage.HTM")
```

システム変数およびセット

送られるファイルが存在する場合やタイムアウトでない場合、OKシステム変数に1が代入されます。その他の場合は0になります。

WEB SEND HTTP REDIRECT

WEB SEND HTTP REDIRECT (url {; *})

引数	型		説明
url	文字	⇨	新しいURL
*	演算子	⇨	指定されている場合 = URLは翻訳されない 省略されている場合 = URLは翻訳される

説明

WEB SEND HTTP REDIRECTコマンドを使用すると、URLをほかのURLへ転送することができます。

引数 url は、リクエストをリダイレクトすることができる新しいURLを格納しています。この引数がファイルへのurlである場合、**WEB SEND HTTP REDIRECT** ("/MyPage.HTM") のようにファイルへの参照を保持していなければなりません。

このコマンドは同じメソッド内にあるデータを送るコマンド (WEB SEND FILE、WEB SEND BLOBなど) より優先されます。

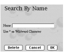
また、このコマンドを使用すると、リクエストを他のWebサーバーへリダイレクトすることも可能です。

4DはURLの特殊文字を自動的にエンコードします。* を渡すと、4Dはそれらをエンコードしません。

このコマンドが送信するHTTPステータスコードは**302: Moved Temporarily**です。"301 Moved Permanently"ステータスを送信したい場合は、WEB SET HTTP HEADERを使用して、*"X-STATUS"フィールドに"301"*を設定します。

例題

このコマンドを使用して、4D上でスタティックページを用いてカスタムリクエストを実行します。以下の要素をスタティックなHTMLページに設定するとします。



注: POSTアクション "/4DCGI/rech" は、テキストエリアと**OK**と**Cancel**ボタンに連携されています。

On Web Connectionデータベースメソッドに以下のコードを挿入します。

```
Case of
   :($1="/4DCGI/rech") // 4DがこのURLを受信したら
 // OKボタンが使用され、 'name' フィールドにValueがある場合
      If((bOK="OK") & (name#""))
 // 同じメソッド内のはるか下に置かれた検索コードを
 // 実行するためにURLを変更する
         WEB SEND HTTP REDIRECT("/4dcgi/rech?"+name)
      Else
 // そうでなければ、始めのページに戻る
         WEB SEND HTTP REDIRECT("/page1.htm")
      End if
      ...
   :($1="/4DCGI/rech?@") // URLがリダイレクトされたら
      ... // 検索コードをここに入れる
End case
```

WEB SEND RAW DATA

WEB SEND RAW DATA (data {; *})

引数	型		説明
data	BLOB	⇒	送るHTTPデータ
*	演算子	⇒	チャンクして送る

説明

WEB SEND RAW DATAコマンドを使用すると、4D Webサーバはチャンクが可能な"生"HTTPデータを送ります。

引数 data はHTTP応答の2つのパートを含みます。つまり、ヘッダとボディです。データは事前にフォーマットされずにサーバから送られます。ただし、応答ヘッダとボディの有効性を確認するため、4Dはそれらに対して基本的なチェックを行います。

- ヘッダが不完全またはHTTPプロトコルの仕様に準拠していない場合、4Dはヘッダを変更します。
- HTTPリクエストが不完全な場合、4Dは不足している情報を追加します。例えば、リクエストをリダイレクトしたい場合、以下のように記述します。

```
HTTP/1.1 302
Location: http://...
```

以下の行のみを渡すと、

```
Location: http://...
```

4DがHTTP/1.1 302 を追加して、そのリクエストを完成します。

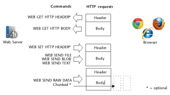
オプション引数を用いると、応答を"チャンク" して送るよう指定できます。応答をいくつかのチャンクに切り分けると、全体の長さを認識せずに、サーバが応答を送る際に役に立ちます (例えば、応答が生成されていない場合) 。すべてのHTTP/1.1互換ブラウザはチャンクされた応答を受け入れます。
引数 * を渡すと、Webサーバは、必要に応じて、応答ヘッダに transfer-encoding: chunked フィールドを自動的に取り入れます (応答ヘッダを手動で処理することも可能です) 。チャンクされたオプションのシンタックスを考慮するために、応答の残りの部分がフォーマットされます。 チャンクされた応答は、1つのヘッダと未定義中図の"チャンク"ボディを含みます。

同じメソッド内で実行された**WEB SEND RAW DATA**(data;*) に続くすべての**WEB SEND RAW DATA**ステートメントは応答の一部と見なされます (それらが引数 * を含んでいるかどうかに関係なく) 。メソッドの実行が終了した際、サーバはチャンクを送ることを停止します。

注: Webクライアントが HTTP/1.1をサポートしていない場合、4Dは応答をHTTP/1.0互換フォーマットへ変換します (送られたデータはチャンクされません) 。ただし、この場合、必ずしも期待している結果を得られるとは限りません。そのため、WebブラウザがHTTP/1.1をサポートしているかどうかをチェックして適切な応答を送ることをお勧めします。これを実行するには、以下のメソッドを使用します。

```
C_BOOLEAN($0)
ARRAY TEXT(arFields;0)
ARRAY TEXT(arValues;0)
WEB GET HTTP HEADER(arFields;arValues)
$0:=False
If(Size of array(arValues)>=3)
   If(Position("HTTP/1.1";arValues{3})>0)
      $0:=True // ブラウザーがHTTP/1.1をサポートすればTrueを返す
   End if
End if
```

新しいWEB GET HTTP BODYコマンド や "Webサーバ" テーマの他のコマンドと組み合わせると、このコマンドは4Dデベロッパが利用できるツールの範囲を完成させます。これは、新旧のHTTP接続の処理を全体的にカスタマイズするためのものです。これら異なるツールを以下の図に表します。



例題

WEB SEND RAW DATAコマンドを用いたチャンクオプションの使用例を表します。データ (数字のシーケンス) はループされオンザフライで生成された100個のチャンクで送られます。応答のヘッダは、明確に設定されません。**WEB SEND RAW DATA**コマンド が応答のヘッダを自動的に送り、それに transfer-encoding: chunked フィールドを挿入します。それは引数 * が適用されているからです。

```
C_LONGINT($cpt)
C_BLOB($my_blob)
C_TEXT($output)

For($cpt;1;100)
   $output:="["+String($cpt)+"]"
   TEXT TO BLOB($output;$my_blob;UTF8 Text without length)
   WEB SEND RAW DATA($my_blob;*)
End for
```

WEB SEND TEXT

WEB SEND TEXT (htmlText {; noContext})

引数	型	説明
htmlText	テキスト	⇒ Webブラウザへ送られるHTMLテキストフィールド または 変数
noContext	テキスト	⇒ True = 非コンテキストモードへ移動 Falseまたは省略 = 現在のモードのまま

説明

WEB SEND TEXTコマンドを使用して、HTMLにフォーマットされたテキストデータを直接送ります。

引数 htmlText は送るデータを格納しています。4Dは引数の内容をチェックしませんので、HTMLが正確にコード化されているかを確認してください。

テキスト内での4D変数と 4DSCRIPT 型タグへの参照は全て解析されます。

type 引数を省略した場合、4Dは送信されたデータが "text/html" 型であると自動的に判断します。この場合、コマンドは WEB SEND BLOB コマンドを使用して "text/html" 型のBLOBを送信したのと同一の挙動をします。
また、type 引数を使用して送信したいテキストのMIME型を指定することもできます。サポートされているMIME型に関しての詳細な情報は、WEB SEND BLOB コマンドの詳細を参照して下さい。

例題

```
C_TEXT($content_t)
C_BLOB($blob_x)
$content_t:="<html><head></head><body>"
$content_t:=$content_t+String(Current time)
$content_t:=$content_t+"</body></html>"

  // 以下の二行を
TEXT TO BLOB($content_t;$blob_x;UTF8 Text without length)
WEB SEND BLOB($blob_x;"text/html")

  // 一行で置き換えることができます。
WEB SEND TEXT($content_t)
```

WEB SET HOME PAGE

WEB SET HOME PAGE (homePage)

引数	型		説明
homePage	文字	→	ページの名前またはページへのHTMLアクセスパス または、""でカスタムホームページを送らない

説明

WEB SET HOME PAGEコマンドを使用して、現在のWebプロセス用のカスタムホームページを変更します。

定義されたページはWebプロセスとリンクしています。そのため、例えば、接続されているユーザーごとに異なるホームページを定義することができます。このページは、静止画でも動画を含んでいても構いません。

HTMLホームページの名前、またはそのページのHTMLアクセスパスを引数 homePage に渡します。
現在のWebプロセスのホームページとして homePage の使用を停止するには、homePage に空の文字列 ("") を渡して**WEB SET HOME PAGE**コマンドを実行します。

注: Webサーバーのデフォルトホームページはデータベース設定ダイアログボックスで定義できます。

WEB SET HTTP HEADER

WEB SET HTTP HEADER (header|fieldArray {; valueArray})

引数	型		説明
header|fieldArray	テキスト, テキスト配列	→	リクエストHTTPヘッダーを格納したフィールドまたは変数、またはHTTPヘッダーフィールド
valueArray	テキスト配列	→	HTTPヘッダーフィールドコンテンツ

説明

WEB SET HTTP HEADERコマンドを使用して、4DからWebブラウザーへ送信されるHTTPヘッダーにフィールドを設定します。このコマンドはWebプロセスでのみ機能します。
このコマンドで "Cookie" を管理することが可能です。

このコマンドでは、2つのシンタックスが利用可能です。

- **1番目のシンタックス**

  WEB SET HTTP HEADER (header)

  設定したいテキストタイプ (変数またはフィールド) のHTTPヘッダーフィールドを 引数 fields に渡します。
  このシンタックスを使用すれば、"HTTP/1.0 200 OK\r\nSet-Cookie: C=HELLO" のようにヘッダーを記述することができます。Windows と Mac OS上では、ヘッダーフィールドは、CR または CR+LF (キャリッジリターン + ラインフィード) シーケンスで区切られてなければなりません。4Dがそのレスポンスをフォーマットします。

  カスタム "Cookie" の例を示します。

```
C_TEXT($vTcookie)
$vTcookie:="Set-Cookie: USER="+String(Abs(Random))+"; PATH=/"
WEB SET HTTP HEADER($vTcookie)
```

  注: コマンドは引数 header として、リテラルテキストを受け入れません。4Dの変数またはフィールドでなくてはなりません。

- **2番目のシンタックス**

  WEB SET HTTP HEADER (fieldArray; valueArray)

  2つのテキスト配列 fieldArray と valueArray を通して、HTTPヘッダーは定義されます。以下のようにヘッダーを記述します

```
fieldArray{1}:="X-VERSION"
fieldArray{2}:="X-STATUS"
fieldArray{3}:="Set-Cookie"

valueArray{1}:="HTTP/1.0"*
valueArray{2}:="200 OK"*
valueArray{3}:="C=HELLO"
```

  * 最初の2つのアイテムはレスポンスの最初の行です。これらを入力する際は、配列の1番目と2番目の要素でなければなりません。ただし、これらを省略して以下のコードのみを記述することも可能です (4Dがヘッダーフォーマットを処理します) 。

```
fieldArray{1}:="Set-Cookie"
valueArray{1}:="C=HELLO"
```

  X-VERSIONとX-STATUSを指定しないと、自動的にHTTP/1.0 200 OKが設定されます。

Server、DateとContent-Lengthのフィールドは常に4Dによって設定されます。

HTTPヘッダーのシンタックスに関する詳細は、http://www.w3c.org のRFC (Request For Comments) を参照してください。

WEB SET OPTION

WEB SET OPTION (selector ; value)

引数	型		説明
selector	倍長整数	➡	オプションコード
value	倍長整数, テキスト	➡	オプション値

説明

WEB SET OPTIONコマンドは4D Webサーバーの機能に関する様々なオプションの値を変更します。

selector引数にはWeb Serverテーマの定数のうちひとつを指定し、valueに新しい設定値を渡します:

定数	型	値	コメント
Web Character set	倍長整数	17	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: データベースに接続するブラウザーとの通信に (ローカルモード4Dと4D Serverの) Webサーバーが使用する文字セット。デフォルト値はOSの言語に依存します。この引数はデータベース設定でも設定できます。 **値**: 取りうる値は、文字セットに関連するデータベースの動作モードによります。 • Unicode モード: アプリケーションがUnicodeモードで動作している場合、この引数に渡す値は文字セット識別子です。(*MIBEnum*倍長整数または文字列。以下のアドレスを参照: http://www.iana.org/assignments/character-sets)。以下は4D Webサーバがサポートする文字セットに対応する識別子のリストです: 4=ISO-8859-1 12=ISO-8859-9 13=ISO-8859-10 17=Shift_JIS 2024=Windows-31J 2026=Big5 38=euc-kr 106=UTF-8 2250=Windows-1250 2251=Windows-1251 2253=Windows-1253 2255=Windows-1255 2256=Windows-1256 **注**: IANAに定義されていない特別な文字セット (1258=x-mac-japanese) を使用することができます。これはWindows上ではコードページ10001に、Mac上ではkTextEncodingMacJapaneseにマップされています。 • *ASCII 互換モード*: 0: Western European 1: Japanese 2: Chinese 3: Korean 4: User-defined 5: Reserved 6: Central European 7: Cyrillic 8: Arabic 9: Greek 10: Hebrew 11: Turkish 12: Baltic
Web HTTP Compression Level	倍長整数	50	**スコープ**: *ローカルWebサーバー* **2セッション間で設定を保持**: No **説明**: 4D HTTPサーバーで使用されるすべての圧縮されたHTTP通信 (クライアントのリクエスト、サーバーのレスポンス、WebおよびWebサービス) の圧縮レベル。このセレクターを使用すれば圧縮率を犠牲にして実行速度を速めるか、速度を優先して圧縮率を高めるかを選択できます。値の選択は交換するデータのサイズやタイプに基づきます。value引数に1から9までの値を渡します。1は速度優先、9は圧縮率優先です。また-1を渡して圧縮速度と圧縮率の妥協点を指定できます。デフォルトの圧縮レベルは1 (速度優先) です。 **とりうる値**: 1 から 9 (1 = 速度優先, 9 = 圧縮優先) または -1 = 最適
Web HTTP Compression Threshold	倍長整数	51	**スコープ**: ローカルHTTPサーバー **2セッション間で設定を保持**: No **説明**: 最適化モードの内部的な4D Webサービス通信フレームワークにおいて、圧縮を行わないリクエストサイズの敷居値を設定できます。この設定は、小さなデータ交換時に圧縮を行うことによる、マシンの時間の浪費を避けるために有効です。 **とりうる値**: 任意の倍長整数値。valueにはバイト単位でサイズを渡します。デフォルトで、圧縮の敷居値は1024バイトに設定されています。
Web HTTPS Port ID	倍長整数	39	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: ローカルモード4Dおよび4D ServerのWebサーバがSSLによるセキュアな接続 (HTTPSプロトコル) で使用するTCPポート番号。HTTPS ポート番号はデータベース設定の"Web/設定"ページで指定できます。デフォルト値は443 (標準ポート番号) です。value引数にTCP Port Numbersテーマの定数を渡すこともできます。 **とりうる値**: 0~65535
Web Inactive process timeout	倍長整数	78	**スコープ**: ローカルWebサーバー **2セッション間で値を保持**: No. しかしHTTPサーバーを再起動しただけの場合は保持されます。 **説明**: セッション管理のために使用されるプロセスのタイムアウトを設定します。タイムアウト後、プロセスは終了されます。 **取りうる値**: 倍長整数 (分) **デフォルト値**: 480分 (= 8時間) (0を渡すとデフォルト値に設定されます)
Web Inactive session timeout	倍長整数	72	**スコープ**: ローカルWebサーバー **2セッション間で値を保持**: No. しかしHTTPサーバーを再起動しただけの場合は保持されます。 **説明**: セッション管理のために使用されるcookieのタイムアウトを設定します。 **取りうる値**: 倍長整数 (分) **デフォルト値**: 480分 (= 8時間) (0を渡すとデフォルト値に設定されます)

Web IP Address to listen	倍長整数	16	**スコープ**: 4D ローカル, 4D Server
			2セッション間で設定を保持: Yes
			説明: ローカルモード4Dおよび4D ServerのWebサーバがHTTPリクエストを受信するIPアドレス。デフォルトで、特定のアドレスは定義されていません (value=0)。この引数はデータベース設定で設定できます。
			Web IP Address to listenセレクターは、コンパイルして4D Volume Desktopを組み込んだ4D Webサーバで役立ちます (この場合デザインモードへのアクセス手段がありません)。
			value引数には16進数のIPアドレスを渡します。つまり、"a.b.c.d"のようなIPアドレスを指定するには、以下のようなコードを作成します:
			`C_LONGINT($addr)`
			`$addr:=($a<<24)|($b<<16)\|($c<<8)|$d`
			`SET DATABASE PARAMETER(Web IP Address to listen;$addr)`

Web Keep session	倍長整数	70	**スコープ**: ローカルWebサーバー **2セッション間で設定を保持**: No. しかしHTTPサーバー再起動後も設定は有効。 **説明**: 4Dによる自動セッション管理モード (Webセッション管理) の有効/無効を設定する。 **取りうる値**: 1 (有効) / 0 (無効) **デフォルト値**: v13で作成されたデータベースでは1、変換されたデータベースでは0。このモードはリモートモードで一時的なコンテキストの再利用メカニズムも有効にする点に留意してください。このメカニズムに関する詳細はWebサーバー設定を参照してください。
Web Log Recording	倍長整数	29	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: ローカルモード4Dまたは4D ServerのWebサーバーが受け取るWebリクエストの記録を開始または停止します。デフォルト値は0 (リクエストを記録しない) です。 Web リクエストのログは"logweb.txt"という名前のテキストファイルに保存されます。このファイルは自動でストラクチャファイルと同階層のLogsフォルダー内に作成されます。このファイルのフォーマットは、渡した値により決定されます。Webログファイルフォーマットに関する詳細はWebサイトに関する情報を参照してください。 このファイルはデータベース設定の"Web/ログ"ページからも有効にできます。 **とりうる値**: 0 = 記録しない (デフォルト), 1 = CLFフォーマットで記録, 2 = DLFフォーマットで記録, 3 = ELFフォーマットで記録, 4 = WLFフォーマットで記録 **警告**: フォーマット3および4はカスタムフォーマットであり、記録する内容を事前にデータベース設定で定義しなければなりません。事前定義せずにこれらのフォーマットを使用した場合、ログファイルは作成されません。
Web Max Concurrent Processes	倍長整数	18	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: ローカルモード4Dならびに4D ServerのWebサーバーでサポートされる、任意のタイプの同時Webプロセス上限数を厳密に設定。この上限数 (マイナス1) に達した場合、4Dはそれ以上プロセスを作成しなくなり、HTTPステータス503 (Service Unavailable) をすべての新しいリクエストに返します。 このパラメーターにより、同時に行われる非常に膨大な数のリクエストやコンテキスト作成に関する過大な要求の結果として、4D Webサーバが飽和状態になることを防ぐことができます。このパラメーターはデータベース設定でも設定できます。 理論上、Webプロセスの最大数は次の計算式の結果になります: 使用可能メモリ/Webプロセスのスタックサイズ。別の解決策は、ランタイムエクスプローラに表示されるWebプロセス情報を監視する方法です。つまり現在のWebプロセス数およびWebサーバの開始以降に達した最大数を監視します。 **値**: 10から32 000までの任意の数。デフォルト値は100。
Web Max sessions	倍長整数	71	**スコープ**: ローカルWebサーバー **2セッション間で設定を保持**: No. しかしHTTPサーバー再起動後も設定は有効。 **説明**: 4Dの自動セッション管理下のセッション上限数。設定した上限に達すると、もっとも古いセッションが閉じられます。 **取りうる値**: 倍長整数値 **デフォルト値**: 100 (0を渡すとデフォルト値が設定されます)
Web Maximum requests size	倍長整数	27	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: Webサーバに処理を許可する受信HTTPリクエスト (POST) の最大サイズ (バイト単位)。デフォルト値は2,000,000 (2MBより少し少ない値) です。最大値 (2,147,483,648) を渡すと、実際上は制限がなくなります。 この制限は、受信するリクエストが大きすぎるためにWebサービスが飽和してしまうことを回避するために使用します。リクエストがこの制限に達すると、4D Webサービスはリクエストを拒否します。 **とりうる値**: 500,000～2,147,483,648.
Web Session cookie name	倍長整数	73	スコープ: ローカルWebサーバー 2セッション間で設定を保持: No. しかしHTTPサーバー再起動後も設定は保持される。 説明: 4Dの自動セッション管理機能で使用されるcookieのname属性値。 取りうる値: テキスト デフォルト値: "4DSID" (空文字を渡すとデフォルト値が設定されます。)

注 (v13.2): selector 引数に使用できるコードが追加されました。将来のバージョンの4Dではこれらのコードのための定数が追加される予定です。v13.2では数値を指定することで使用できます:

- selector = 15: ローカルモードの4Dおよび4D ServerのWebサーバーで使用されるTCPポート番号を設定/取得します。設定値は倍長整数で、デフォルト値は80です。ここで設定/取得される値はデータベース設定の"Web/設定"ページで見ることができるものです。value 引数にはTCP Port Numbersテーマの定数を指定できます。このセレクターは4D Volume Desktopが統合されたコンパイル済み4D Webサーバーアプリケーションでポート番号を変更する場合に有効です。TCPポート番号についてはWebサーバー設定を参照してください。
- selector = 81: セッションCookieの"domain"フィールドの値を設定/取得します (テキスト)。
- selector = 82: セッションCookieの"path"フィールドの値を設定/取得します (テキスト)。
  セレクター81と82はセッションCookieのスコープを制御するために使用します:
  - セレクター81で value を "*.4d.jp" に設定すると、ユーザーエージェントに対しリクエストがドメイン"*.4d.jp" (* はワイルドカード文字) に送信されるときのみ、Cookieを送出するよう指示できます。
  - セレクター82で value を "/4DACTION" に設定すると、ユーザーエージェントに対しパスが "/4DACTION" で始まるリクエストにのみ、Cookieを送出するよう指示できます。

WEB SET ROOT FOLDER

WEB SET ROOT FOLDER (rootFolder)

引数	型		説明
rootFolder	文字	→	Webサーバルートフォルダのパス名

説明

WEB SET ROOT FOLDERコマンドを使用して、デフォルトのルートフォルダを変更します。そのなかで4DはWebサーバがリクエストしたHTMLを探します。

このコマンドはデータベース設定に設定されているデフォルトのルートフォルダを考慮にいれません。このフォルダに関する詳細は接続セキュリティを参照してください。

HTMLシンタックス (URLタイプ) 、またはシステムシンタックス (絶対パス) のいずれかでルートフォルダの場所を表します。

- HTMLシンタックス: 使用しているプラットフォームに関係なく、フォルダの名前をスラッシュ ("/") で区切ります。
- システムシンタックス: 現在のプラットフォームのシンタックスを考慮した絶対パス名 ("ロングネーム") 。
  - (Mac OS) Disk:Applications:myserv:folder
  - (Windows) C:\Applications\myserv\folder

Notes:

- 新しいルートフォルダを考慮に入れるために、Webサーバの再起動が必要です。
- Get 4D folderコマンドを使用すると、何時でも現在のルートフォルダの場所を探すことができます。

無効なパス名を指定すると、OS Fileマネージャエラーが生成されます。ON ERR CALLメソッドでこのエラーを検知できます。エラーメソッド内から警告またはメッセージを表示する場合、ブラウザ側で表示されます。

WEB START SERVER

WEB START SERVER
このコマンドは引数を必要としません

説明

WEB START SERVERコマンドは、実行中の4Dアプリケーション上でWebサーバを起動します。結果、データベースはイントラネットやインターネット上で公開されます。

システム変数およびセット

Webサーバの起動が成功すると、OK に1が代入されます。その他の場合は、OK に 0 (ゼロ) が代入されます。例えば、TCP/IPネットワークのプロトコルが正確に構成されていないと、OK に 0 が代入されます。

WEB STOP SERVER

WEB STOP SERVER
このコマンドは引数を必要としません

説明

WEB STOP SERVERコマンドは、実行中の4Dアプリケーション上で、Webサーバを停止します 。Webサーバが既に起動している場合、すべてのWeb接続は停止し、すべてのWebプロセスが終了します。
Webサーバが起動していない場合、コマンドは何も行いません。

WEB Validate digest

WEB Validate digest (userName ; password) -> 戻り値

引数	型		説明
userName	テキスト	→	ユーザの名前
password	テキスト	→	ユーザのパスワード
戻り値	ブール	←	True=認証はOK、False=認証は失敗

説明

WEB Validate digestコマンドを使用して、Webサーバに接続しているユーザのID情報 (名前とパスワード) の有効性をチェックします。必ずダイジェストモードのWeb認証のコンテキストで、On Web Authenticationデータベースメソッド でこのコマンドを使用します (接続セキュリティ を参照)。

引数 userName と password には、ローカルに格納されているユーザ識別情報を渡します。コマンド はこの情報を使用して、Webブラウザによって送られた情報と同等な値を生成します。

値が同じである場合、コマンドはTrueを返します。その他の場合は、Falseを返します。

このメカニズムを使用して、Webサーバへアクセスする自身の安全なシステムをプログラミングによって維持、そして管理することができます。

Note: ブラウザがダイジェスト認証をサポートしていない場合、エラーが返されます (認証エラー) 。

例題

ダイジェストモードで On Web Authenticationデータベースメソッド を使用します:

```
  ` On Web Authentication Database Method
C_TEXT($1;$2;$5;$6;$3;$4)
C_TEXT($user)
C_BOOLEAN($0)
$0:=False
$user:=$5
  `セキュリティに関する理由のため、@を含む名前を拒否する
If(WithWildcard($user))
   $0:=False
  `WithWildカードメソッドは、"On Web Authentication データベースメソッド" の節に記述されています
Else
   QUERY([WebUsers];[WebUsers]User=$user)
   If(OK=1)
      $0:=WEB Validate digest($user;[WebUsers]password)
   Else
      $0:=False `ユーザーが存在しません
   End if
End if
```

_o_SET CGI EXECUTABLE

_o_SET CGI EXECUTABLE (url1 {; url2})

引数	型		説明
url1	文字	⇒	アクセスURL
url2	文字	⇒	アクセスURL

Command disabled

このコマンドは4D v13で廃止されました。このコマンドを呼び出すとエラー33 ("メソッドまたは関数が実装されていません") が生成されます。

_o_SET WEB DISPLAY LIMITS

_o_SET WEB DISPLAY LIMITS (numberRecords {; numberPages {; picRef}})

引数	型		説明
numberRecords	**倍長整数**	⇒	***** 無効 *****
numberPages	**倍長整数**	⇒	***** 無効 *****
picRef	**倍長整数**	⇒	***** 無効 *****

このコマンドは使用できません

このコマンドは4D v13より使用できなくなりました。このコマンドを実行するとエラー33 (コマンドまたは関数が実装されていません) が生成されます。

_o_SET WEB TIMEOUT

_o_SET WEB TIMEOUT (timeout)

引数	型		説明
timeout	倍長整数	⇒	*** 無効 ***

このコマンドは使用できません

このコマンドは4D v13より使用できなくなりました。このコマンドを実行するとエラー33 (コマンドまたは関数が実装されていません) が生成されます。

_o_Web Context

_o_Web Context -> 戻り値

引数	型		説明
戻り値	ブール	←	*** 無効 ***

このコマンドは使用できません

このコマンドは4D v13より使用できなくなりました。このコマンドを使用するとエラー33 (メソッドまたは関数が実装されていません) が生成されます。

Webサービス (クライアント)

- Webサービス (クライアント) コマンド
- WEB SERVICE AUTHENTICATE
- WEB SERVICE CALL Updated 14.0
- WEB SERVICE Get info
- WEB SERVICE GET RESULT
- WEB SERVICE SET OPTION
- WEB SERVICE SET PARAMETER

Webサービス (クライアント) コマンド

バージョン2003以降、4Dは “Webサービス” をサポートしています。プログラムを使用して、自身のデータベースから直接的にWebサービスを公開したり、クライアントとして使用したりできます。Webサービスは、ネットワーク上で公開された一連の関数です。ネットワークに接続、またはWebサービスと互換のあるアプリケーションがこれらの関数を呼び出したり、使用したりします。Webサービスは、すべてのタイプのタスクを実行します。例えば運送会社による荷物の配送管理、e-commerce、市場価格のモニタリングなどです。
Webサービスのコンセプトと操作に関する詳細は、Design Referenceマニュアルを参照してください。

4Dでは、Webサービスウィザードを使用して、簡単にWebサービスの利用手続きを行うことができます。ほとんどの場合ウィザードを使用するだけで、Webサービス利用できます。ただし、あるメカニズムをカスタマイズしたい場合、4DのクライアントSOAPコマンドを使用しなければなりません。

この節では、外部のWebサービス (**クライアント側**) の利用手続きの際に使用するコマンドに関して説明します。Webサービス (サーバ側) を公開するために使用するコマンドについての詳細は、テーマを参照してください。

Note: 従来からの決まりにより、サーバ側とクライアント側の間でコマンド(と定数) の名前を区別するために、“SOAP” と “Webサービス” という用語が、それぞれに使用されています。この2つのコンセプトは同じテクノロジーを参照しています。

WEB SERVICE AUTHENTICATE

WEB SERVICE AUTHENTICATE (name ; password {; authMethod} {; *})

引数	型	説明
name	文字	→ ユーザの名前
password	文字	→ ユーザのパスワード
authMethod	倍長整数	→ 認証メソッド 0または省略された場合 = 指定されない、1 = BASIC、2 = DIGEST
*	演算子	→ 渡された場合、プロキシによる認証

説明

WEB SERVICE AUTHENTICATEコマンドは、クライアントアプリケーションの認証を必要とするWeb サービスの使用を可能にします (シンプルな認証) 。BASICとDIGEST メソッド、そしてプロキシの存在がサポートされています。

注: BASICとDIGEST認証のメソッドに関する詳細は接続セキュリティを参照してください。

引数 name と password には、必須となるID情報 (ユーザ名とパスワード) を渡します。WEB SERVICE CALLコマンドによってこの情報はコード化され、Webサービスへ送られるHTTPリクエストに追加されます。従ってWEB SERVICE CALLコマンドを呼び出す前に**WEB SERVICE AUTHENTICATE**コマンド を呼び出す必要があります。

オプション引数 authMethod を用いて、次のWEB SERVICE CALLコマンド を呼び出すために使用する認証メソッドを指定します。そのためには、以下の値の1つを渡します。

- 2 = DIGEST認証メソッドを使用する。
- 1 = BASIC認証メソッドを使用する。
- 0 (または引数を省略) = 適切なメソッドを使用する。この場合、4Dは認証メソッドを決定するために追加のリクエストを発行します。

引数 * を渡すと、認証情報をHTTPプロキシへ送ることを指定します。Webサービスクライアントと Webサービスの間で認証を必要とするプロキシが存在する際、必ずこの設定を実行しなければなりません。Webサービス自体が認証されている場合、ダブル認証が必要となります (例題を参照) 。

認証情報は各リクエスト後にデフォルトで0にリセットされます。そのため**WEB SERVICE AUTHENTICATE**コマンドを使用してから各WEB SERVICE CALLコマンドを使用します。しかしWEB SERVICE SET OPTIONコマンドのオプションを使用すれば、この情報を一時的に保持するのは可能です。この場合、**WEB SERVICE AUTHENTICATE**コマンドをつど実行せずに、各WEB SERVICE CALLコマンドを実行します。

認証が失敗すると、SOAPサーバはエラーを返します。このエラーはWEB SERVICE Get infoコマンドを使用して確認できます。

例題

プロキシの後ろに配置されているWebサービスによる認証:

```
  //DIGESTモードでのWeb Serviceへの認証
 WEB SERVICE AUTHENTICATE("SoapUser";"123";2)
  //デフォルトモードでのプロキシへの認証
 WEB SERVICE AUTHENTICATE("ProxyUser";"456";*)
 WEB SERVICE CALL(...)
```

WEB SERVICE CALL

WEB SERVICE CALL (accessURL ; soapAction ; methodName ; nameSpace {; complexType {; *}})

引数	型		説明
accessURL	文字	⇒	WebサービスへのアクセスURL
soapAction	文字	⇒	SOAPActionフィールドの内容
methodName	文字	⇒	メソッドの名前
nameSpace	文字	⇒	名前空間
complexType	倍長整数	⇒	複合タイプの設定 (省略された場合、シンプルタイプ)
*	演算子	⇒	接続を終了しない

説明

WEB SERVICE CALLコマンドは、HTTPリクエストを送ることでWebサービスを呼び出すために使用します。このリクエストには、事前にWEB SERVICE SET PARAMETERコマンドを使用して作成したSOAPメッセージが含まれています。

一連のWEB SERVICE SET PARAMETERコマンド呼び出しは、新しいリクエストを作成します。**WEB SERVICE CALL**コマンドの実行はまた、以前に呼び出したWebサービスの結果を消去し、新しい結果に置き換えます。

accessURL には、Webサービスへアクセスできる完全なURLを渡します (このURLをWebサービスを説明するWSDLファイルのURLと混同しないでください) 。

- **保護モードでのアクセス(SSL)**: SSLを用いて、保護モードでWebサービスを使用したい場合、http://の代わりとしてURLの前にhttps://を渡します。この設定により、保護モードでの自動接続が可能になります。なお、このコマンドはサーバー認証を使用することができます(HTTP SET CERTIFICATES FOLDER コマンドを参照して下さい)。この認証が有効でない(失効しているか無効である)場合、OKシステム変数は0に設定され、エラー901 "Invalid server certificate" が返されます。このエラーは ON ERR CALL コマンドに実装されたエラーハンドリングメソッドにより割り込み可能です。

soapAction には、リクエストのSOAPActionフィールドの内容を渡します。このフィールドは一般的に"ServiceName#MethodName"値を含みます。

methodName には、(Webサービスに属する) 実行したいリモートメソッドの名前を渡します。

namespace には、SOAPリクエストに使用するXML名前空間を渡します。XML名前空間に関する詳細は、4DのDesign Referenceマニュアルを参照してください。

オプション引数 complexType は、(WEB SERVICE SET PARAMETERとWEB SERVICE GET RESULTコマンドを使用して定義された) 送信および受信するWebサービス引数の設定を示します。

引数 complexType の値は、Webサービスの公開モード (DOC または RPC、4DのDesign Referenceマニュアルを参照してください) とその引数により異なります。
complexType にはWeb Services (Client)テーマにある以下の定数の1つを必ず渡します:

定数	型	値
Web Service Dynamic	倍長整数	0
Web Service Manual	倍長整数	3
Web Service Manual In	倍長整数	1
Web Service Manual Out	倍長整数	2

それぞれの定数は、Webサービス"設定"に対応しています。設定は公開モード (RPC/DOC) と引数のタイプ (入出力、シンプルまたは複合) の組み合わせを表します。

Note: 引数 "input" または "output" の特徴は、プロキシメソッド/Webサービスの視点から評価されるということを覚えておいてください。

- 引数 "input" はプロキシメソッドに、そしてWebサービスへ渡される値です。
- 引数 "output" は、Webサービスから、そしてプロキシメソッドから返されます (一般的に$0経由) 。

可能となる設定と対応する定数のすべてを以下のリストに示します:

	Input引数	
Output引数	**シンプル**	**複合**
シンプル	Web Service Dynamic (RPCモード)	Web Service Manual In (RPCモード)
複合	Web Service Manual Out (RPCモード)	Web Service Manual (RPCまたはDOCモード)

使用する設定に応じて、このリクエストの内容をフォーマットするかは開発者に任されています。

そのため、以下で説明する5つの設定を実装することができます。すべての場合において、4DはWebサービスへ送られるSOAPリクエストやそのエンベロープのフォーマット処理を行います。

Note: 複合XMLタイプであるにも関わらず、4Dはシンプルタイプとしてデータの配列を処理します。

RPCモード、シンプルinputとoutput

最も使いやすいのはこの設定です。この場合、引数 complexType には定数 Web Service Dynamic が含まれるかこの引数が省略されます。

送られた引数と受け取った応答は、前処理されることなく直接的に処理できます。

WEB SERVICE GET RESULTコマンドの例題を参照してください。

RPCモード、複合inputとシンプルoutput

この場合、引数 complexType には定数 Web Service Manual In が含まれます。この設定ではWebサービスに、WEB SERVICE SET PARAMETERコマンドを使用してBLOBの形でXMLソースの要素を必ず "手動で" 渡さなければなりません。

最初のBLOBを有効なXML要素としてフォーマットするかしないかを決定するのは、ユーザ次第です。最初の要素として、このBLOBには、最終リクエストの <body> 要素の最初に現れる "child" 要素が含まれなければなりません。

例題

```
 C_BLOB($1)
 C_BOOLEAN($0)
 WEB SERVICE SET PARAMETER("MyXMLBlob";$1)
 WEB SERVICE CALL("http://my.domain.com/my_service";"MySoapAction";"TheMethod";"http://my.namespace.com/";Web
Service Manual In)
 WEB SERVICE GET RESULT($0;"MyOutputVar";*)
```

RPCモード、シンプルinputと複合output

この場合、引数complexTypeには、定数Web Service Manual Outが含まれます。それぞれの引数outputはBLOBに保存されているXML要素の形でWebサービスから返されます。WEB SERVICE GET RESULTコマンド. を使用してこの引数を取り出します。そして、受け取ったBLOBの内容を4Dの XMLコマンドを使用して解析します。

例題

```
 C_BLOB($0)
 C_BOOLEAN($1)

 WEB SERVICE SET PARAMETER("MyInputVar";$1)
 WEB SERVICE CALL("http://my.domain.com/my_service";"MySoapAction";"TheMethod";"http://my.namespace.com/";Web
Service Manual Out)

 WEB SERVICE GET RESULT($0;"MyXMLOutput";*)
```

RPCモード、複合inputとoutput

この場合、complexType 引数にはWeb Service Manual定数を指定します。それぞれのinputとoutput引数は、先に説明した2つの設定に従い、XML要素の形でBLOBに格納しなければなりません。**例題**

```
 C_BLOB($0)
 C_BLOB($1)

 WEB SERVICE SET PARAMETER("MyXMLInputBlob";$1)
 WEB SERVICE CALL("http://my.domain.com/my_service";"MySoapAction";"TheMethod";"http://my.namespace.com/";Web
Service Manual)
 WEB SERVICE GET RESULT($0;"MyXMLOutput";*)
```

DOCモード

DOC Webサービスのプロクシ呼び出しメソッドは、複合型INPUT OUTPUT引数を使用するRPC Webサービスのプロクシ呼び出しメソッドと同様です。
これら2つの設定の唯一の違いは、送受信するBLOB引数のXMLの内容レベルにあります。4Dから見ると、SOAPリクエストの構築と送信は同じものです。

例題

```
 C_BLOB($0)
 C_BLOB($1)

 SET WEB SERVICE PARAMETER("MyXMLInput";$1)
 WEB SERVICE CALL("http://my.domain.com/my_service";"MySoapAction";"TheMethod";"http://my.namespace.com/";Web
Service Manual)
 WEB SERVICE GET RESULT($0;"MyXMLOutput";*)
```

注: DOC Webサービスの場合、引数として渡される文字列の値 (上記の"MyXMLInput" と "MyXMLOutput") は重要ではありません。空の文字列を渡すことさえできます。実際、これらの値はXMLドキュメントを含むSOAPリクエストでは使用されません。にもかか わらずこれらの引数は必須です。

DOCモード (または複合タイプのRPCモード) で公開されたWebサービスを使用するには、以下に従うことをお勧めします:

- Webサービスウィザードを使用してプロクシメソッドを生成する。
  プロクシメソッドは以下のように呼び出されます: $XMLresultBlob:=$DOCproxy_Method($XMLparamBlob)
- オンラインテストにより、Webサービスに送信するSOAPリクエストの内容について習熟する (例えばhttp://soapclient.com/soaptest.html)。このタイプのツールは、WebサービスのWSDLに基づき、HTMLテストフォームを生成するために使用されます。
- 生成されたXMLの<body>の第一子要素をコピーします。
- 実際の引数値をXMLコードに置くことを可能にするメソッドを記述する。このコードは$XMLparamBlob BLOBに置かれます。
- レスポンスを解析するには、オンライテストを使用するか、返される要素を定義したWSDLを参照します。

オプションの * 引数を使用して呼び出しを最適化できます。この引数を指定すると、コマンドはその実行後に、処理により使用される接続を閉じません。この場合、次の**WEB SERVICE CALL**は * 引数が渡されていれば同じ接続を再利用します。接続を閉じるには、**WEB SERVICE CALL**コマンドを * 引数なしで呼び出します。このメカニズムにより、同じサーバ上の複数の異なるWebサービスを連続して呼び出すような処理が高速化されます。特にWAN環 境 (例えばインターネット経由) の場合に顕著です。この設定はWebサーバの"keep-alive"設定に基づきます。この設定では通常接続ごとの最大リクエスト数を設定し、リクエ ストを拒否することもあります。一連の**WEB SERVICE CALL**リクエストがこの最大数の制限に達した場合、またはkeep-alive接続が許可されていない場合、4Dは新しい接続を作成します。

システム変数およびセット

リクエストが正しくルーティングされ、Webサービスがそれを受け入れると、システム変数OKは1に設定されます。そうでなければ0に設定され、エラーが生成されます。

WEB SERVICE Get info

WEB SERVICE Get info (infoType) -> 戻り値

引数	型		説明
infoType	倍長整数	→	取得する情報
戻り値	文字	↩	最新のSOAPエラーに関する情報

説明

WEB SERVICE Get infoコマンドは、リモートのWebサービスに送信された最新のSOAPリクエストの 実行中に生成されたエラーについての情報を返します。このコマンドは一般的にON ERR CALLコマンドでインストールされたエラー処理メソッド内から呼び出します。

infoType 引数には、取得したい情報を指定します。Web Services (Client)テーマの以下の定数を渡します:

定数	型	値	コメント
Web Service Detailed Message	倍長整数	1	エラーの説明メッセージ。メインのエラータイプに応じてメッセージは異なります。 - 9910 (Soap fault): SOAPエラーの原因が返されます (例: "リモートメソッドが存在しません")。 - 9911 (Parser fault): XMLドキュメント中のエラーの個所が返されます。 - 9912 (HTTP fault): - HTTPエラーが [300-400] の場合 (問題がリクエストされたドキュメントの場所に関連する場合)、リクエストURLの新しい場所が返されます。 - 他のHTTPエラーコードの場合、<body>が返されます。 - 9913 (Network fault): ネットワークエラーの原因が返されます (例: "ServerAddress: DNS lookup failure") - 9914 (Internal fault): 内部エラーの原因が返されます。
Web Service Error Code	倍長整数	0	(4Dが定義した) 主たるエラーコード。このコードはErrorシステム変数にも返されます。 返される可能性のあるコードは以下のとおりです: 9910: Soap fault (参照 Web Service Fault Actor) 9911: Parser fault 9912: HTTP fault (参照 Web Service HTTP Error code) 9913: Network fault 9914: Internal fault.
Web Service Fault Actor	倍長整数	3	エラーの原因 (SOAPプロトコルにより返される、メインエラーが9910の場合に使用)。 - バージョンが合わない - 引数解釈エラー (必須として定義された引数をサーバが解釈できない) - 送信側側のエラー - 受信側のエラー - 未知のエンコーディング
Web Service HTTP Error code	倍長整数	2	HTTPエラーコード (メインエラーが9912の場合に使用)。

情報が利用できない場合、空の文字列が返されます。特に最後のSOAPリクエストがエラーを生成しなかった場合。

WEB SERVICE GET RESULT

WEB SERVICE GET RESULT (returnValue {; returnName {; *}})

returnValue	変数	⇐	Webサービスから返された値
returnName	文字	⇒	取得する引数の名前
*		⇒	メモリを解放

説明

WEB SERVICE GET RESULTコマンドは、行われた処理の結果としてWebサービスから返された結果を取得するために使用します。

注: このコマンドはWEB SERVICE CALLコマンドの後に使用しなければなりません。

returnValue 引数はWebサービスから送り返された値を受け取ります。この引数には4D変数を渡します。この変数は通常、プロクシメソッドから返される値に対応する$0です。しかし中間変数を使用することもできます (プロセス変数のみ)。

注: 使用する4Dの変数や配列は、事前にコンパイラまたは配列コマンドを使用して宣言しなければなりません。

オプションのreturnName 引数を使用して、取得する引数の名前を指定します。ほとんどのWebサービスは1つの値しか返さないので、通常この引数は必要ありません。

オプションの * 引数はプログラムに、リクエストの処理に使用したメモリを解放するよう指示します。この引数はWebサーバーから返された最後の値を取得してから指定しなければなりません。

例題

Webサービスがある都市の時間を返すとします。Webサービスに渡す引数は都市名と国コードです。Webサービスは対応する時間を返します。呼び出しプロクシメソッドは以下のようになります:

```
 C_TEXT($1)
 C_TEXT($2)
 C_TIME($0)

 WEB SERVICE SET PARAMETER("city";$1)
 WEB SERVICE SET PARAMETER("country_code";$2)

 WEB SERVICE
CALL("http://www.citiesoftheworld.com/WS";"WSTime#City_time";"City_time";"http://www.citiesoftheworld.com/namespace/default")

 If(OK=1)
    WEB SERVICE GET RESULT($0;"return";*)
 End if
```

WEB SERVICE SET OPTION

WEB SERVICE SET OPTION (option ; value)

引数	型		説明
option	倍長整数	→	設定するオプションのコード
value	倍長整数, テキスト	→	オプションの値

予備的なお知らせ

このコマンドは上級Webサービスユーザのためにデザインされています。使用するかどうかは任意です。

説明

WEB SERVICE SET OPTIONコマンドを使用して、WEB SERVICE CALLコマンドを使用して次回呼び出されるSOAPリクエストで使用されるさまざまなオプションを設定できます。
設定するオプションの数だけこのコマンドを呼び出します。

option 引数には設定するオプションの番号、value 引数にはそのオプションの新しい値を渡します。これらの引数にはWeb Services (Client)テーマの以下の定義済み定数を使用できます:

定数	型	値	コメント
Web Service display auth dialog	倍長整数	4	value = 0 (ダイアログボックスを表示しない) または 1 (ダイアログボックスを表示する) このオプションはWEB SERVICE CALLコマンド実行時の認証ダイアログボックスの表示を管理します。デフォルトでダイアログボックスは表示されません。通常WEB SERVICE AUTHENTICATEコマンドを使用して認証を行わなければなりません。しかし認証ダイアログボックスを表示してユーザに認証情報を入力させたい場合、このオプションを使用します。value に1を渡すとダイアログを表示、0を渡すと表示しません。ダイアログボックスはWebサービスが認証を要求する場合のみ表示されます。
Web Service HTTP Compression	倍長整数	6	value = Web Service Compression このオプションは、SOAPリクエストによる4Dアプリケーション間のデータ交換を高速化する目的で、内部的な圧縮メカニズムを有効にするために使用します。WEB SERVICE SET OPTION(Web Service HTTP Compression; Web Service Compression) ステートメントをWebサービスの4Dクライアント上で実行すると、クライアントから送信される次のSOAPリクエストのデータは、4D SOAPサーバーに送信される前に、標準のHTTPメカニズム (リクエストの内容に応じて"gzip" または "deflate") を使用して圧縮されます。サーバーはリクエストを解凍・解析し、自動で同じメカニズムを使用して応答します。WEB SERVICE SET OPTIONコマンドの呼出し直後のリクエストのみに影響します。すなわち圧縮を使用したいリクエスト毎にこのコマンドを呼び出す必要があります。デフォルトで4DはWebサービスHTTPリクエストを圧縮しません。 **注:** • このメカニズムは11.3以前の4D SOAPサーバーへのリクエストでは利用できません。 • この機能を最適化するために、リクエストの圧縮を行う閾値や圧縮率を設定する追加のオプションがあります。これらのオプションを設定するにはSET DATABASE PARAMETERコマンドを使用します。
Web Service HTTP Timeout	倍長整数	1	value = 秒単位で指定するクライアント側のタイムアウト クライアント側のタイムアウトは、サーバが返答しない場合のWebサービスクライアント側の待ち時間です。この時間経過後、クライアントはセッションを閉じ、リクエストは失われます。 このタイムアウトはデフォルトで180秒です。特定の理由 (ネットワークの状態、Webサービスの仕様等) でこの値を変更できます。
Web Service reset auth settings	倍長整数	5	value = 0 (情報を消去しない) または 1 (情報を消去する) このオプションを使用して、4Dにユーザの認証情報 (ユーザ名とパスワード、認証メソッド等) を記憶させ、それを再利用するかどうかを指定できます。デフォルトでこの情報はWEB SERVICE CALLコマンドを呼び出すたびに消去されます。value に0を渡すと情報は保持され、1を渡すと消去されます。0を渡した場合、情報はセッションの間保持されます。
Web Service SOAP Header	倍長整数	2	value = SOAPリクエストのヘッダとして挿入するXMLルート要素参照 このオプションを使用して、WEB SERVICE CALLコマンドで生成されるSOAPリクエストにヘッダを挿入できます。デフォルトでSOAPリクエストは特定のヘッダを持っていません。しかしWebサービスによっては、例えば識別情報を管理するために、ヘッダを要求することがあります。
Web Service SOAP Version	倍長整数	3	value = Web Service SOAP_1_1 または Web Service SOAP_1_2 このオプションで、リクエストで使用するSOAPプロトコルのバージョンを指定できます。Web Service SOAP_1_1 定数を value に渡すとバージョン1.1が、Web Service SOAP_1_2 を渡すとバージョン1.2が指定されます。

オプションを呼び出す順番は重要ではありません。同じ option が複数回設定された場合は、最後の呼び出しで設定された値が有効になります。

例題 1

SOAPリクエストにカスタマイズしたヘッダーを挿入する:

```
  // XML参照を作成
 C_TEXT(vRootRef;vElemRef)
 vRootRef:=DOM Create XML Ref("RootElement")
 vxPath:="/RootElement/Elem1/Elem2/Elem3"
 vElemRef:=DOM Create XML element(vRootRef;vxPath)
  // 参照を使用してSOAPヘッダーを更新する
 WEB SERVICE SET OPTION(Web Service SOAP Header;vElemRef)
```

例題 2

SOAPプロトコルのバージョン1.2を使用する:

```
 WEB SERVICE SET OPTION(Web Service SOAP Version;Web Service SOAP_1_2)
```

WEB SERVICE SET PARAMETER

WEB SERVICE SET PARAMETER (name ; value {; soapType})

引数	型		説明
name	文字	→	SOAPリクエストに含める引数の名前
value	変数	→	引数の値を格納する4D変数
soapType	文字	→	引数のSOAPタイプ

説明

WEB SERVICE SET PARAMETERコマンドはクライアントのSOAPリクエストで引数の定義を行います。リクエスト中の引数毎にこのコマンドを呼び出します (このコマンドの呼び出し回数は引数の数に応じます)。

nameには、SOAPリクエストに現れる引数の名前を渡します。

valueには、引数に渡す値を格納した4D変数を渡します。プロクシメソッドの場合、この変数は通常、プロクシメソッドが呼び出されるときに渡される引数$1, $2, $3, 等です。しかし中間変数を使用することもできます。

Note: それぞれの4D変数や配列はコンパイラや配列テーマのコマンドで事前に宣言しなければなりません。

デフォルトで4Dはvalueの内容に基づき、name 引数にもっとも適切なSOAPタイプを使用します。指定されたタイプはリクエストに含まれます。
しかし、引数のSOAPタイプを指定したい時があります。この場合、オプションのsoapType 引数に以下の文字列の一つを渡すことができます (主たるデータタイプ):

soapType	説明
string	文字列
int	倍長整数
boolean	ブール
float	32-bit 実数
decimal	小数付き実数
double	64-bit 実数
duration	年, 月, 日, 時間, 分, 秒による期間, 例えば P1Y2M3DT10H30M
datetime	ISO8601フォーマットの日付と時間、例 2003-05-31T13:20:00
time	時間、例 13:20:00
date	日付、例 2003-05-31
gyearmonth	年と月 (グレゴリオ暦)、例 2003-05
gyear	年 (グレゴリオ暦)、例 2003
gmonthday	月と日 (グレゴリオ暦)、例 --05-31
gday	日 (グレゴリオ暦)、例 ---31
gmonth	月 (グレゴリオ暦)、例 --10--
hexbinary	16進表現の値
base64binary	BLOB
anyuri	Uniform Resource Identifier (URI)、例 http://www.company.com/page
qname	有効な XML 名 (名前空間とローカル部分)
notation	記法属性

Note: XMLデータタイプに関する詳細は、URL http://www.w3.org/TR/xmlschema-2/ を参照してください。

例題

この例題は2つの引数を定義します:

```
C_TEXT($1)
C_TEXT($2)
WEB SERVICE SET PARAMETER("city";$1)
WEB SERVICE SET PARAMETER("country";$2)
```

Webサービス (サーバ)

- Webサービス (サーバ) コマンド
- SOAP DECLARATION
- SOAP Get info
- SOAP request
- SOAP SEND FAULT

Webサービス (サーバ) コマンド

4DによるWebサービスの公開は、メソッドプロパティの設定で行います。ほとんどの場合、この処理だけでWebサービスの公開を行うことができます。しかし特定のメカニズムをカスタマイズしたい場合、例えば配列を使用したい場合などは、この章のコマンドを使用します。

この章では4DでWebサービスを公開するために使用するコマンドを説明します (**サーバ側**)。Webサービスに関する一般的な情報やWebサービスを利用するために使用するコマンドに関する説明は、を参照してください。

Note: 慣例に従って、用語"SOAP" と "Web Service" はコマンドと定数で、サーバとクライアントとの識別を行うために使用されていますこれら2つのコンセプトは同じテクノロジを参照しています。

SOAP DECLARATION

SOAP DECLARATION (variable ; type ; input_output {; alias})

引数	型		説明
variable	変数	→	入出力SOAP引数を参照する変数
type	倍長整数	→	引数が指す4Dの型
input_output	倍長整数	→	1 = SOAP入力, 2 = SOAP出力
alias	文字	→	SOAP交換の間 この引数用に公開される名前

説明

SOAP DECLARATION コマンドを使用して、Webサービスとして公開された4Dメソッドで使用される引数の型を明示的に宣言できます。

メソッドがWebサービスとして公開されると、標準の引数$0, $1... $nが外部へのWebサービス引数の定義に使用されます (特にWSDLファイル内で)。SOAPプロトコルは、引数が明示的に命名されていることを要求します。4Dは"FourD_arg0, FourD_arg1 ... FourD_argN"をデフォルトで使用します。

このデフォルトの動作は、以下の理由で問題となる場合があります:

- $0 や $1, $2を配列として宣言できません。ゆえにポインタを使用する必要がありますが、この場合、値の型はWSDLファイル生成時に分かっていなければなりません。
- 入出力引数の名前を変更したい、あるいは変更しなければならない時があります。
- XML構造とDOM参照を引数として使用したい場合があります。
- 32 KBを超えるサイズの値を返したい時があります (非Unicodeモードのテキストサイズの上限)。
- 最後に、この動作だとRPC呼び出しごとに複数の値を返すことができません ($0に)。

SOAP DECLARATION コマンドを使用すればこれらの制限から解放されます。このコマンドを入出力引数毎に実行し、名前と型を割り当てます。

Note: SOAP DECLARATION コマンドが使用されていても、依然として4Dの変数と配列をCompiler_Webメソッド内で"コンパイラ"テーマのコマンドを使用して宣言する必要があります。

variableには、Webサービスを呼び出すときに参照される4D変数を渡します。

警告: プロセス変数または4D引数 ($0 から $n) のみを参照できます。ローカルおよびインタープロセス変数は使用できません。

デフォルトで、テキスト型の引数のみが使用できるため、非Unicodeモードのデータベースでは、SOAPサーバのレスポンスは32KBに制限されます。しかしBLOBを使用すれば32KBを超えるレスポンスを返信できます。この制限を超えるには、SOAP DECLARATION コマンドを呼び出す前に引数をBLOBとして宣言します (例題 4参照)。

Note: クライアント側では、このタイプのWebサービスを4Dから呼び出した場合、Webサービスウィザードはテキスト型の変数を生成します。これを使用可能にするには、プロクシメソッド内の値を受け取る変数をBLOB型に変更します。

typeには対応する4Dのデータ型を渡します。ほとんどのタイプの4D変数と配列を使用できます。リストプロパティの設定テーマの以下の定義済み定数と、XMLタイプの場合テーマの2つの定数を使用できます:

定数	型	値
Boolean array	倍長整数	22
Date array	倍長整数	17
Integer array	倍長整数	15
Is BLOB	倍長整数	30
Is Boolean	倍長整数	6
Is Date	倍長整数	4
Is Integer	倍長整数	8
Is LongInt	倍長整数	9
Is Real	倍長整数	1
Is String Var	倍長整数	24
Is Text	倍長整数	2
Is Time	倍長整数	11
LongInt array	倍長整数	16
Real array	倍長整数	14
String array	倍長整数	21
Text array	倍長整数	18

定数	型	値
Is DOM reference	倍長整数	37
Is XML	倍長整数	36

input_outputには、処理される引数が"入力" (例. メソッドが値を受け取る場合に対応) または "出力" (例. メソッドが値を返す場合に対応) であるかを示す値を渡します。テーマの以下の定義済み定数を使用できます:

定数	型	値
SOAP Input	倍長整数	1
SOAP Output	倍長整数	2

XML タイプの利用

Is XML と Is DOM reference 定数を使用して、入出力共に変数を"XML structure" と "DOM reference" 型に宣言できます。このタイプの引数が指定されると、引数への処理やエンコードは行われず、データはそのまま送信されます (例題 5参照)。

- 出力引数:
  - Is XML は引数がXML構造であることを示します。
  - Is DOM reference はXML構造のDOM参照であることを示します。この場合、XML構造をSOAPメッセージに挿入することは、DOM EXPORT TO VAR コマンドを実行することと同じです。

Note: 出力引数としてを使用するDOM参照の場合、例えばOn Startupで作成し、アプリケーションの終了時に廃棄されるグローバルな参照を使用することをお勧めします。実際Webサービス内で作成されるDOM参照は、DOM CLOSE XMLで閉じることはできません。これはメソッドの実行が終了して、値が実際にWebサービスクライアントに返されるときにも、XML参照が存在していなければならないためです。Webサービスを複数回呼び出せば、閉じられないDOM参照が複数できることになり、メモリ不足を引き起こすかもしれません。

- 入力引数:
  - Is XML はSOAPクライアントから送信されたXML引数を受け取ることを示します
  - Is DOM reference はSOAPクライアントから送信されたXML引数に対応するXML構造のDOM参照を受け取ることを示します。

- WSDLの変更: これらのXML構造は4Dにより"anyType" 型 (不定) としてWSDL内で宣言されます。XML構造を明確に型宣言したい場合、WSDLファイルを保存して、手作業によりWSDLの<types>部に適切なデータスキーマを追加します。

COMPILER_WEB メソッド

4D変数を参照する入力SOAP引数 (4Dメソッド引数を除く) はまずCOMPILER_WEBプロジェクトメソッド内で宣言されなければなりません。実際Webサービスメソッド内でプロセス変数を使用するには、メソッドが呼び出される前にそれらが宣言されていることを必要とします。COMPILER_WEBプロジェクトメソッドは、存在すれば、SOAPリクエストが受け入れられるたびに呼び出されます。デフォルトでCOMPILER_WEBメソッドは存在しません。明示的に作成する必要があります。

COMPILER_WEBメソッドはWebサーバでも使用されます (参照)。

aliasには、WSDLとSOAP交換時に表示される引数の名前を渡します。

警告: この名前はRPC内でユニークでなければなりません (入力、出力引数両方で)。そうでなければ最後の宣言のみが有効になります。

Note: 引数名は数字で始まっていてはならず、スペースを含むことはできません。さらに互換性の問題を避けるため、アクセント文字などの拡張文字は使用しないことをお勧めします。
alias 引数が省略されると、4Dはデフォルトで変数名または4Dメソッド引数にはFourD_argNを使用します。

Note: SOAP DECLARATION コマンドはWebサービスとして公開されるメソッドに含まれていなければなりません。他のメソッドから呼び出すことはできません。

例題 1

この例題は引数名を指定します:

```
  ` Webサービスメソッド内で
  ` WSDLファイルの生成時とSOAP呼び出し時に、
  ` zipcode が fourD_arg1の代わりに使用されます。
 SOAP DECLARATION($1;Is LongInt;SOAP Input;"zipcode")
```

例題 2

この例題は郵便番号の配列を倍長整数型で受け取ります:

```
  `COMPILER_WEB メソッド
 ARRAY LONGINT(codes;0)

  `Webサービスメソッド
 SOAP DECLARATION(codes;LongInt array;SOAP Input;"in_codes")
```

例題 3

この例題では、引数名を指定せずに、2つの戻り値を参照しています:

```
 SOAP DECLARATION(ret1;Is LongInt;SOAP Output)
 SOAP DECLARATION(ret2;Is LongInt;SOAP Output)
```

例題 4

この例題は、非Unicodeモードのデータベースにおいて、32KBを超える値を返すことを可能にします:

```
 C_BLOB($0)
 SOAP DECLARATION($0;Is Text;SOAP Output)
```

Note タイプ Is Text (Is BLOBではなく)を使用していることに留意してください。これにより引数が正しく処理されます。

例題 5

以下の例題では様々なタイプの宣言を示しています:

```
 ALL RECORDS([Contact])

  `XML構造をContactsのセレクションから構築し、XMLをBLOBに格納
 C_BLOB(ws_vx_xmlBlob)
 getContactsXML(->ws_vx_xmlBlob)
  `XML構造をテキスト変数に取り出す
 C_TEXT(ws_vt_xml)
 ws_vt_xml:=BLOB to text(ws_vx_xmlBlob;UTF8 Text without length)
  `XML構造のDOM参照を取得
 C_TEXT(ws_vt_xmlRef)
 ws_vt_xmlRef:=DOM Parse XML variable(ws_vt_xml)

  `さまざまな宣言をテスト
 SOAP DECLARATION(ws_vx_xmlBlob;Is BLOB;SOAP Output;"contactListsX")
  `XMLは4DによりBase64に変換される

 SOAP DECLARATION(ws_vt_xml;Is Text;SOAP Output;"contactListsText")
  `XMLはテキストに変換される (< > は実体参照になる)

 SOAP DECLARATION(ws_vt_xml;Is XML;SOAP Output;"xmlContacts")
  `XMLはXMLテキストとして渡される

 SOAP DECLARATION(ws_vx_xmlBlob;Is XML;SOAP Output;"blobContacts")
  `XMLはXML BLOBとして渡される

 SOAP DECLARATION(ws_vt_xmlRef;Is DOM reference;SOAP Output;"contactByRef")
  `XML参照を渡す
```

SOAP Get info

SOAP Get info (infoNum) -> 戻り値

引数	型		説明
infoNum	倍長整数	→	取得するSOAP情報のタイプ番号
戻り値	文字	↩	SOAP情報

説明

SOAP Get infoコマンドを使用して、SOAPリクエストに関するさまざま情報を文字列で受け取ることができます。

SOAPリクエストを処理する際、RPC引数のほかに、リクエストに関する追加の情報を得られると便利です。例えばセキュリティのため、On Web Authenticationデータベースメソッド内でこのコマンドを使用して、リクエストされたWebサービスのメソッド名を知ることができます。

知りたいSOAP情報のタイプ番号をinfoNum 引数に渡します。Web Services (Server)テーマで定義済みの以下の定数を使用できます:

定数	型	値	コメント
SOAP Method Name	倍長整数	1	実行されようとしているWebサービスメソッド名
SOAP Service Name	倍長整数	2	メソッドが属しているWebサービスの名前

Note: またセキュリティのため、4Dに送信されるWebサービスリクエストの最大サイズを設定できます。この設定はSET DATABASE PARAMETERコマンドで行います。

SOAP request

SOAP request -> 戻り値

引数	型		説明
戻り値	ブール	↩	True: リクエストはSOAP; そうでなければFalse

説明

SOAP requestコマンドは実行されているコードがSOAPリクエストの一部であればTrueを返します。

このコマンドは、セキュリティの目的でOn Web Authenticationデータベースメソッドで使用し、受信したリクエストがSOAPであるか知ることができます。

SOAP SEND FAULT

SOAP SEND FAULT (faultType ; description)

引数	型		説明
faultType	倍長整数	⇒	1 = クライアント側のエラー, 2 = サーバー側のエラー
description	文字	⇒	SOAPクライアントに送信する、エラーの説明

説明

SOAP SEND FAULTコマンドを使用して、SOAPクライアントにエラーの発生元 (クライアントまたはサーバ) を示すエラーを送ることができます。このコマンドを使用することで、結果を返さなくても、クライアントにエラー示すことができます。

例えば、“Square_root” Webサービスが呼び出された時に負数がクライアントから送信された場合、クライアントによるエラーが検知します。このコマンドを使用してクライアントに、正数が必要であることを通知できます。

サーバ側のエラーとしては、例えば、メソッド実行時のメモリ不足などがあります。

faultTypeにエラーの発生元を渡します。Web Services (Server)テーマの定義済み定数を使用できます:

定数	型	値
SOAP Client Fault	倍長整数	1
SOAP Server Fault	倍長整数	2

descriptionにはエラーの説明を渡します。クライアントの実装が対応していれば、エラーを処理できます。

例題

“Square_root” Webサービスの例題において、負数が渡された場合にリクエストを処理するため、以下のコードを使用できます:

```
SEND SOAP FAULT(SOAP Client Fault;"Positive values required")
```

XML

- XMLユーティリティコマンドの概要
- XML DECODE
- XML GET ERROR
- XML GET OPTIONS
- XML SET OPTIONS
- XSLT APPLY TRANSFORMATION
- XSLT GET ERROR
- XSLT SET PARAMETER

XMLユーティリティコマンドの概要

このテーマには4Dの汎用XMLユーティリティコマンドがまとめられています。XMLオプション、エラー管理およびXSLに特化したコマンドがあります。

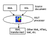
XMLに関する一般的な情報やDOMとSAXの違いなどはXML DOMコマンドの概要を参照してください。

XSL 変換の管理

4DはXSL スタイルシート (eXtended Stylesheet Language) のアプリケーションをサポートしています。XSLランゲージを使用してXMLドキュメントのタグを変更することができます。

XSLランゲージの機能には2つの異なる側面があります:

- **整形**: HTMLのCSSのように、XML要素にスタイルや表示ルールを指定できます。
- **変換**: XMLタグをHTMLなどの異なるタグシステムに変更できます。この機能は**XSLT**として知られています。XSLスタイルシートはドキュメントのXML要素を選択することで全体的に認識し、それらを他の要素に変換します。この機能は例えば異なる種類のドキュメントを同期させる場合に便利です。



Note: 4DはXSL変換を行うためにXalan-C_1_6_0.dllライブラリを使用します。Xalanはフリーウェアの XSLTプロセッサです。詳細はhttp://xml.apache.org/xalan-c/index.htmlを参照してください。

XSLスタイルシートはマニュアルまたは特別なアプリケーションを使用して生成されたテキストドキュメントで、.xsl拡張子が与えられます。XSLランゲージはさまざまな要素や機能を持っていて、動的な変換を行うことができます。このランゲージ関する詳細は例えばhttp://xmlfr.orgを参照してください。

4Dでは既存のXSLスタイルシートを使用してXMLドキュメントを変換できます (XSLT APPLY TRANSFORMATION コマンド)。また4DではXSLT SET PARAMETERコマンドを使用してXSLスタイルシートのパラメタをオンザフライで変更することもできます。

Note: 書き出しダイアログボックスのオプションを使用してXML書き出し時にXDLスタイルシートを使用し、変換されたXMLドキュメントを生成することができます。

SVGとは

SVG (Scalable Vector Graphics) はファイルフォーマットで、ベクタグラフィックをXMLで定義するために使用されます (拡張子は.svg)。SVGのもっとも一般的な利用シーンは統計や地図データの公開です。
これらのファイルはWebブラウザでネイティブに、あるいはプラグインを使用して表示させることができます。4D v11にはSVG描画エンジンが含まれていて、SVGファイルをピクチャフィールドや変数に表示させることができます。DOM EXPORT TO PICTUREコマンドを使用して4DでSVG定義に基づくピクチャを生成できます。またGRAPH コマンドを使用して4Dに統合されたSVGエンジンを利用することもできます。
このフォーマットに関する詳細は以下のWebサイトを参照してください: http://www.w3.org/Graphics/SVG/.

XML DECODE

XML DECODE (xmlValue ; 4Dobject)

引数	型		説明
xmlValue	テキスト	→	XML構造から取得したテキスト型の値
4Dobject	フィールド, 変数	←	変換したXMLの値を受け取る4D変数またはフィールド

説明

XML DECODE コマンドはXML文字列に格納されている値を4D型の値に変換します。変換は以下のルールに基づいて自動で行われます:

値	例	日本語システム上での変換例
数値	<Price>8,5</Price><Price>8.5</Price>	実数: 8.5
ブール	<Double>1</Double> <Double>0</Double> または<Double>true</Double> <Double>false</Double>	ブール: True/False
BLOB		Base64 デコード
ピクチャ		Base64 デコード + BLOB to picture コマンド
日付	2009-10-25T01:03:20+01:00	時間部とタイムゾーンを取り除く: !10/25/2009!
時間	2009-10-25T01:03:20+01:00	日付部とタイムゾーンを取り除く: ?01:03:20?

例題

XMLドキュメントから、属性として格納されているデータを読み込む。
XMLドキュメントの例:

```
<CD  Date="2003-01-01T00:00:00Z" Description="This double CD reissued by EMI  in 1995 combines 4 Stabat mater hymns. One by Rossini interpreted by the  Berlin Symphony Orchestra, directed by Karl Forster. Followed by a work  of Verdi, by the Philharmonic Orchestra, directed by Carlo Maria  Giulini. On the second CD, you will find Francis Poulenc interpreted by  Régine Crespin. This compilation ends with a little-known version, that  of the Polish composer Karol Szymanowski. Polish National Radio Symphony  Orchestra directed by Antoni Wit" Double="true" Duration="7246"  Type="Sacred music" CD_ID="5" Performer="Various" Price="8.5" Title="4  Stabat mater"/>
```

```
 Repeat
    MyEvent:=SAX Get XML node(DocRef)

    Case of
       :(MyEvent=XML Start Element)
          ARRAY TEXT(arrAttrNames;0)
          ARRAY TEXT(arrAttrValues;0)
          SAX GET XML ELEMENT(DocRef;vName;vPrefix;arrAttrNames;arrAttrValues)
          If(vName="CD")
             CREATE RECORD([CD])
             For($i;1;Size of array(arrAttrNames))
                $attrName:=arrAttrNames{$i}
                Case of
                   :($attrName="CD_ID")
                      XML DECODE(arrAttrValues{$i};[CD]CD_ID)
                   :($attrName="Title")
                      [CD]Work:=arrAttrValues{$i}
                   :($attrName="Price")
                      XML DECODE(arrAttrValues{$i};[CD]Price)
                   :($attrName="Date")
                      XML DECODE(arrAttrValues{$i};[CD]Date entered)
                   :($attrName="Duration")
                      XML DECODE(arrAttrValues{$i};[CD]Total_duration)
                   :($attrName="Double")
                      XML DECODE(arrAttrValues{$i};[CD]Double_CD)
                End case
             End for
          End if
          ...
    End case
 Until(MyEvent=XML Start Document)
```

XML GET ERROR

XML GET ERROR (elementRef ; errorText {; row {; column}})

引数	型		説明
elementRef	文字	→	XML要素参照
errorText	変数	←	エラーテキスト
row	変数	←	行番号
column	変数	←	列番号

説明

XML GET ERROR コマンドはerrorText 引数に、elementRef 引数で指定されたXML要素の処理中に検知したエラーの説明を返します。返される情報はXerces.DLLライブラリから提供されるものです。

オプションのrowとcolumn引数はエラーの場所を示します。これらの引数にはエラーが発生した行と、その行の中でのエラーの最初の文字の位置が返されます。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

XML GET OPTIONS

XML GET OPTIONS (elementRef | document ; selector ; value {; selector2 ; value2 ; ... ; selectorN ; valueN})

引数	型		説明
elementRef | document	テキスト, DocRef	→	XMLルート要素参照 または開かれたドキュメントの参照
selector	倍長整数	→	取得するオプション
value	倍長整数	←	オプションの現在値

説明

XML GET OPTIONS コマンドは、カレントセッションおよびカレントユーザで使用されている1つ以上のXMLパラメタの現在値を取得するために使用します。

selectorには取得するオプションを指定する、""テーマの定義済み定数を渡します。オプションの現在の値はvalue 引数に返されます:

定数	型	値	コメント
XML Binary encoding	倍長整数	5	バイナリデータを変換する方法を指定します。 とりうる値は: • XML Base64 (デフォルト値): バイナリデータは単純にBase64に変換される • XML Data URI scheme: バイナリデータはBase64に変換され、"data:;base64"ヘッダが追加される。このフォーマットは主に、ブラウザが自動でピクチャをデコードできるようにするために使用されます。またSVGピクチャの挿入にも必要です。詳細はhttp://en.wikipedia.org/wiki/Data_URI_schemeを参照してください。
XML Date encoding	倍長整数	2	4D日付の変換方法を指定します。例えば日本のタイムゾーンで !2003/01/01! の例で、とりうる値は (時差によりUTCでは日付が異なる場合があります): • XML ISO (デフォルト値): タイムゾーンの指定なしでxs:datetimeフォーマットを使用します。結果は: "2003-01-01"。時間部がSQLなどを使用して4D値に含まれていても、それは失われます。 • XML Local: タイムゾーンを指定してxs:dateフォーマットを使用します。結果は: "2003-01-01 +09:00"。時間部がSQLなどを使用して4D値に含まれていても、それは失われます。 • XML Datetime local: タイムゾーンを指定してxs:dateTime (ISO 8601) フォーマットを使用します。時間部がSQLなどを使用して4D値に含まれている場合、このフォーマットでは時間部が保持されます。結果は: "<Date>2003-01-01T00:00:00 +09:00</Date>"。 • XML UTC: xs:dateフォーマットを使用します。結果は: "2003-01-01Z"。時間部がSQLなどを使用して4D値に含まれていても、それは失われます。 • XML Datetime UTC: xs:dateTime (ISO 8601)フォーマットを使用します。時間部がSQLなどを使用して4D値に含まれている場合、このフォーマットでは時間部が保持されます。結果は: "<Date>2003-01-01T00:00:00Z</Date>".
XML Indentation	倍長整数	4	XMLドキュメントのインデントを指定します。 とりうる値: • XML With indentation (デフォルト値): ドキュメントはインデントされる • XML No indentation: ドキュメントはインデントされない。内容は一行中に置かれます。
XML Picture encoding	倍長整数	6	(Base64にエンコードされる前に) ピクチャの変換の方法を指定します。 とりうる値: • XML Convert to PNG (デフォルト値): Base64にエンコードされる前に、ピクチャはPNGに変換されます。 • XML Native codec: Base64にエンコードされる前に、ピクチャは最初のネイティブなストレージCODECに変換されます。SVGピクチャをエンコードするためにこれらのオプションを使用しなければなりません (XML SET OPTIONSコマンドの例題参照)。
XML String encoding	倍長整数	1	4D文字列を要素値に変換する方法を指定します。これはXMLでエスケープ文字の利用が必須である属性の変換には影響しません。 とりうる値: • XML With escaping (デフォルト値): 4D文字列をXML要素値に変換する際、文字の置き換えを行います。テキスト型のデータは自動で解析され、禁止されている文字 (<&>") はXML実体参照 (&<> "") に置き換えられます。 • XML Raw data: 4D文字列は生データとして送信されます。4Dはエンコードや解析を行いません。4Dの値は可能であればXMLフラグメントに変換され、ターゲット要素の子要素として挿入されます。値をXMLフラグメントとして扱えない場合、新しいCDATAノードに生データとして挿入されます。
XML Time encoding	倍長整数	3	4Dの時間を変換する方法を指定します。例: ?02/00/46? (日本時間)。エンコーディングは時刻を表すか時間を表すかにより異なります。 時刻の場合: • XML Datetime UTC: UTC (Universal Time Coordinated) で表現された時刻。UTCへの変換は自動です。結果: "<Duration>0000-00-00T17:00:46Z</Duration>". • XML Datetime local: 時刻は4Dエンジンが実行されているマシンの時差を使用して表現されます。結果: "<Duration>0000-00-00T02:00:46+09:00</Duration>". • XML Datetime local absolute (デフォルト値): 時刻は時差なしで表現されます。値は変更されません。結果: "<Duration>0000-00-00T02:00:46</Duration>". 時間の場合: • XML Seconds: 00:00:00からの経過秒数。時間をあらわすため、値は変更されません。結果: "<Duration>7246</Duration>". • XML Duration: XML Schema Part 2に準拠した時間表現。時間をあらわすため、値は変更されません。結果: "<Duration>PT02H00M46S</Duration>".

XML SET OPTIONS

XML SET OPTIONS (elementRef | document ; selector ; value {; selector2 ; value2 ; ... ; selectorN ; valueN})

引数	型		説明
elementRef | document	テキスト, DocRef	→	XMLルート要素参照、または開かれたドキュメント参照
selector	倍長整数	→	設定するオプション
value	倍長整数	→	オプションの値

説明

XML SET OPTIONS コマンドを使用して、第一引数に渡されたXML構造のXMLオプションの値を変更できます。

このコマンドは"ツリー"タイプ (DOM) や"ドキュメント" タイプ (SAX) のXML構造に適用できます。第一引数にはルート要素参照 (elementRef)、あるいは開かれたSAXドキュメント (document) を渡します。

このコマンドで設定されるオプションは、4DからXMLの方向でのみ利用されます (4DへのXML値の読み込みには効果ありません)。以下のコマンドがこのオプションを使用します:

- DOM SET XML ELEMENT VALUE
- DOM SET XML ATTRIBUTE
- SAX ADD XML ELEMENT VALUE

selector に変更するオプションを渡し、valueにオプションの新しい値を渡します。必要なだけselector/valueの組を渡すことができます。

"" テーマの以下の定数を使用しなければなりません:

定数	型	値	コメント
XML Binary encoding	倍長整数	5	バイナリデータを変換する方法を指定します。 とりうる値は: • XML Base64 (デフォルト値): バイナリデータは単純にBase64に変換される • XML Data URI scheme: バイナリデータはBase64に変換され、"data:;base64"ヘッダが追加される。このフォーマットは主に、ブラウザが自動でピクチャをデコードできるようにするために使用されます。またSVGピクチャの挿入にも必要です。詳細はhttp://en.wikipedia.org/wiki/Data_URI_schemeを参照してください。
XML Date encoding	倍長整数	2	4D日付の変換方法を指定します。例えば日本のタイムゾーンで !2003/01/01! の例で、とりうる値は (時差によりUTCでは日付が異なる場合があります): • XML ISO (デフォルト値): タイムゾーンの指定なしでxs:datetimeフォーマットを使用します。結果は: "2003-01-01"。時間部がSQLなどを使用して4D値に含まれていても、それは失われます。 • XML Local: タイムゾーンを指定してxs:dateフォーマットを使用します。結果は: "2003-01-01 +09:00"。時間部がSQLなどを使用して4D値に含まれていても、それは失われます。 • XML Datetime local: タイムゾーンを指定してxs:dateTime (ISO 8601) フォーマットを使用します。時間部がSQLなどを使用して4D値に含まれている場合、このフォーマットでは時間部が保持されます。結果は: "<Date>2003-01-01T00:00:00 +09:00</Date>"。 • XML UTC: xs:dateフォーマットを使用します。結果は: "2003-01-01Z"。時間部がSQLなどを使用して4D値に含まれていても、それは失われます。 • XML Datetime UTC: xs:dateTime (ISO 8601)フォーマットを使用します。時間部がSQLなどを使用して4D値に含まれている場合、このフォーマットでは時間部が保持されます。結果は: "<Date>2003-01-01T00:00:00Z</Date>".
XML Indentation	倍長整数	4	XMLドキュメントのインデントを指定します。 とりうる値: • XML With indentation (デフォルト値): ドキュメントはインデントされる • XML No indentation: ドキュメントはインデントされない。内容は一行中に置かれます。
XML Picture encoding	倍長整数	6	(Base64にエンコードされる前に) ピクチャの変換の方法を指定します。 とりうる値: • XML Convert to PNG (デフォルト値): Base64にエンコードされる前に、ピクチャはPNGに変換されます。 • XML Native codec: Base64にエンコードされる前に、ピクチャは最初のネイティブなストレージCODECに変換されます。SVGピクチャをエンコードするためにこれらのオプションを使用しなければなりません (XML SET OPTIONSコマンドの例題参照)。
XML String encoding	倍長整数	1	4D文字列を要素値に変換する方法を指定します。これはXMLでエスケープ文字の利用が必須である属性の変換には影響しません。 とりうる値: • XML With escaping (デフォルト値): 4D文字列をXML要素値に変換する際、文字の置き換えを行います。テキスト型のデータは自動で解析され、禁止されている文字 (<&>") はXML実体参照 (&<> "") に置き換えられます。 • XML Raw data: 4D文字列は生データとして送信されます。4Dはエンコードや解析を行いません。4Dの値は可能であればXMLフラグメントに変換され、ターゲット要素の子要素として挿入されます。値をXMLフラグメントとして扱えない場合、新しいCDATAノードに生データとして挿入されます。
XML Time encoding	倍長整数	3	4Dの時間を変換する方法を指定します。例：?02/00/46? (日本時間)。エンコーディングは時刻を表すか時間を表すかにより異なります。 時刻の場合: • XML Datetime UTC: UTC (Universal Time Coordinated) で表現された時刻。UTCへの変換は自動です。結果: "<Duration>0000-00-00T17:00:46Z</Duration>". • XML Datetime local: 時刻は4Dエンジンが実行されているマシンの時差を使用して表現されます。結果: "<Duration>0000-00-00T02:00:46+09:00</Duration>". • XML Datetime local absolute (デフォルト値): 時刻は時差なしで表現されます。値は変更されません。結果: "<Duration>0000-00-00T02:00:46</Duration>". 時間の場合: • XML Seconds: 00:00:00からの経過秒数。時間をあらわすため、値は変更されません。結果: "<Duration>7246</Duration>". • XML Duration: XML Schema Part 2に準拠した時間表現。時間をあらわすため、値は変更されません。結果: "<Duration>PT02H00M46S</Duration>".

Notes:

- XML LocalおよびXML Datetime local 値はUTC (Universal Time Coordinated)で表現された日付を提供しません。日付は変更されず、時差が付加されます。このフォーマットは変換してその逆変換をおこなうような場合に便利です。
- XML UTCおよびXML Datetime UTC 値はフォーマット上は先と同じですが、UTCで表現されます。相互の互換性のために、このフォーマットを優先的に使用すべきです。値は変更されません。

例題

SVG ピクチャの挿入:

```
XML SET OPTIONS($pictElemRef;XML Binary encoding;XML Data URI scheme)
XML SET OPTIONS($pictElemRef;XML Picture encoding;XML Native codec)
DOM SET XML ATTRIBUTE($pictElemRef;"xlink:href";PictVar)
```

XSLT APPLY TRANSFORMATION

XSLT APPLY TRANSFORMATION (xmlSource ; xslSheet ; result {; compileSheet})

引数	型		説明
xmlSource	文字, BLOB	→	XMLソースドキュメントの名前またはアクセスパス、またはXMLソースを含むBLOB
xslSheet	文字, BLOB	→	XSLスタイルシートドキュメントの名前または アクセスパス、または XSLスタイルシートを含むBLOB
result	文字, BLOB	→	XSL変換の結果を受け取るドキュメントの名前または アクセスパス、または XSL変換の結果を受け取るBLOB
compileSheet	ブール	→	True = XSL変換を最適化 Falseまたは省略 = 最適化しない、コンパイルされたXSLファイルがあれば削除する

説明

XSLT APPLY TRANSFORMATION コマンドはXSL変換をXMLを含むドキュメントまたはBLOBに適用し、結果のドキュメントまたはBLOBを生成します。このコマンドのスコープはカレントプロセスです。

Note: XSL変換 (XSLT) に関する詳細はXMLユーティリティコマンドの概要を参照してください。

このコマンドは3つのBLOBまたは文字列引数を使用します。

警告: このコマンドは変数またはフィールドのみを引数として受け取ります。

文字列を渡すと、ドキュメントを指定したことになります。この場合、(データベースストラクチャと同階層にある) ドキュメント名、またはドキュメントへのフルパスを渡します。
同じ呼び出し内で異なる型の引数を渡すことはできません。

- xmlSource 引数は変換するXMLソースが含まれていなければなりません。コマンドはXMLコードが有効か検証します。
- xslSheet 引数にはXSL変換に使用するXSLスタイルシートが含まれていなければなりません。このスタイルシートはマニュアルまたは特別なアプリケーションを使用して生成されます。コマンドはXMLコードが有効か検証します。
- result 引数にはXSL変換の結果を受け取るドキュメント名またはBLOBが渡されなければなりません。存在しないドキュメント名を渡した場合、4Dは自動でそれを作成します。ドキュメントが書き込みアクセスで既に開かれていると、エラーが生成されます。

コマンドはXMLソースを解析し、XSLスタイルシートの指示に従い変換します。XSLT SET PARAMETERコマンドが事前に使用されていれば、コマンドは定義されたパラメタを置き換えます。変換の結果はresultドキュメントまたはBLOBに書き込まれます。

オプションのcompileSheet 引数を使用して、XSLT変換を最適化できます。特に連続して同じXSLスタイルシートを用いて変換を行うアプリケーションで有効です。compileSheet 引数にTrueが設定されると、XSLファイルxslSheetはコマンドの最初の呼び出しで解析され、コンパイルののちメモリに格納されます。同じXSLスタイルシートを引き続き呼び出す場合、コマンドはコンパイルされたファイルを直接使用します (変更されない限り)。これにより処理が高速化されます。最適化はxsl:importにより読み込まれるファイル中で実行された変更は考慮しません。XSLにより参照されるファイルが更新された場合、新しいXSLファイルを強制的に再コンパイルしなければなりません。これを行うにはcompileSheet 引数にFalseを渡すかこの引数を省略して、このコマンドを再度呼び出します。

例題

XSLT SET PARAMETER コマンドの例題参照

システム変数およびセット

変換が正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

XSLT GET ERROR

XSLT GET ERROR (errorText {; row {; column}})

引数	型		説明
errorText	変数	←	エラーテキスト
row	変数	←	行番号
column	変数	←	列番号

説明

XSLT GET ERROR コマンドはerrorText 引数に、カレントプロセスでXSL変換中に最後に発生したエラーの説明を返します。返される情報はXerces.DLLライブラリから提供されるものです。

オプションのrowとcolumn引数はXSLドキュメント中のエラーの場所を示します。これらの引数にはエラーが発生した行と、その行の中でのエラーの最初の文字の位置が返されます。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

XSLT SET PARAMETER

XSLT SET PARAMETER (paramName ; paramValue)

引数	型	説明
paramName	文字 ⇒	XSLスタイルシート中のパラメタ名
paramValue	文字 ⇒	変換されたドキュメントで使用するパラメタの値

説明

XSLT SET PARAMETER コマンドはXSLT APPLY TRANSFORMATION コマンドと共に使用しなければなりません。このコマンドは、XMLドキュメントのXSL変換が開始される際に、XSLスタイルシートに置かれた変数値の定義を可能にします。このコマンドを使用して、XSLT APPLY TRANSFORMATIONの直前に4Dプロセス由来の値をXSLスタイルシートに挿入できます。

Notes:

- XSL変換 (XSLT) に関する詳細はXMLユーティリティコマンドの概要を参照してください。
- このコマンドのスコープはカレントプロセスです。関連するXSLT APPLY TRANSFORMATION コマンドと同じプロセスで呼ばれなければなりません。

paramNameには置き換え対象のXSL変数引数の名前を渡します。この引数はXSLスタイルシート中で$toreplaceのように書かれていなければなりません。しかし$文字はparamName中では必要ありません。例えばXSLファイルにインストラクション <xsl:template match=$myvar> が置かれているとき、paramNameには"myvar"だけを渡します。

paramValueには、変換されたファイル中でXSL変数の代わりに挿入したい値を渡します。例えば先の例題でparamValueに"title"を渡すと、XSL変換により <xsl:template match="title"> インストラクションが使用されます ("title"要素が件名としてスタイルルールに使用されます)。
値のタイプが文字列の場合、4Dシンタックスのダブルクォートに加え、値をシングルクォートで囲まなければなりません (例えば "'myvalue'")。

Note: XSLランゲージに関する説明はWebサイト、たとえばhttp://xml.orgなどを参考にしてください。

XSLスタイルシートに複数のパラメタを渡すには、XSLT SET PARAMETER コマンドを必要なだけ呼び出します。同じプロセス内でXSLT APPLY TRANSFORMATIONが呼び出されるまでパラメタはスタックされます。XSLT APPLY TRANSFORMATIONが実行されると、スタックされたパラメタは自動で削除されます。

例題

以下の例題では2つのXSLパラメタをスタックし、mysheet.xslスタイルシートを使用してドキュメントmydoc.xmlをHTMLファイルに変換します:

```
XSLT SET PARAMETER("varstyle";"'bold'")
XSLT SET PARAMETER("varcolor";"'blue'")
$xmldoc:="mydoc.xml"
$xslsheet:="mysheet.xsl"
$htmldoc:="mydoc.html"
XSLT APPLY TRANSFORMATION($xmldoc;$xslsheet;$htmldoc)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

XML DOM

- XML DOMコマンドの概要
- DOM Append XML child node
- DOM Append XML element
- DOM CLOSE XML
- DOM Count XML attributes
- DOM Count XML elements
- DOM Create XML element
- DOM Create XML element arrays
- DOM Create XML Ref
- DOM EXPORT TO FILE
- DOM EXPORT TO VAR
- DOM Find XML element
- DOM Find XML element by ID
- DOM Get first child XML element
- DOM Get last child XML element
- DOM Get next sibling XML element
- DOM Get parent XML element
- DOM Get previous sibling XML element
- DOM Get Root XML element
- DOM GET XML ATTRIBUTE BY INDEX
- DOM GET XML ATTRIBUTE BY NAME
- DOM GET XML CHILD NODES
- DOM Get XML document ref
- DOM Get XML element
- DOM GET XML ELEMENT NAME
- DOM GET XML ELEMENT VALUE
- DOM Get XML information
- DOM Insert XML element
- DOM Parse XML source
- DOM Parse XML variable
- DOM REMOVE XML ATTRIBUTE
- DOM REMOVE XML ELEMENT
- DOM SET XML ATTRIBUTE
- DOM SET XML DECLARATION
- DOM SET XML ELEMENT NAME
- DOM SET XML ELEMENT VALUE

XML DOMコマンドの概要

4DにはXML (eXtensible Markup Language) データを含むオブジェクトを解析するための一連のコマンドがあります。

XMLランゲージについて

XMLランゲージはデータ交換の標準です。タグの使用に基づき、また交換されるデータや構造の明確な定義を可能にします。XMLファイルはテキストフォーマットファイルです。内容はアプリケーションにより読み込まれ、解析されます。多くのアプリケーションがいまやこのフォーマットをサポートしています。XMLに関する詳細は例えば、http://xml.org や http://www.w3.org などのサイトを参照してください。

XMLサポートのために、4DはApache Foundationにより開発されたXerces.dllライブラリを使用しています。4DはXML version 1.0をサポートしています。

Note: 4Dは書き出し/読み込みエディタを使用したXMLフォーマットでのデータ読み込みと書き出しをサポートしています。

DOMと SAX

このテーマのコマンドはDOM接頭辞が付いています。4DはDOMとSAX接頭辞がつく2つのセットのXMLコマンドを提供しています。DOM (Document Object Model) と SAX (Simple API XML) はXML用の2つの異なる解析モデルです。

- DOMモードでは、XMLソースを解析してその構造 (ツリー) をメモリに構築します。このため、それぞれの要素へのアクセスはとても速く行えます。しかしツリー構造の全体がメモリに格納されるため、大きなXMLドキュメントの処理ではメモリ不足を招き、結果エラーが生成されるかもしれません。
- SAXモードでは、メモリにツリー構造を構築しません。このモードではソースの解析時に (要素の開始と終了のような) イベントが生成されます。このモードではどんなサイズのXMLドキュメントでも、利用可能なメモリ量にかかわらず解析することができます。SAXコマンドは"XML SAX"テーマにまとめられています。詳細はの節を参照してください。

XML標準に関する情報は以下のサイトをご覧ください:
http://www.saxproject.org/?selected=event と
http://www.w3schools.com/xml/

DOMによるXMLの作成、開く、及び閉じる

4DのDOMコマンドで作成、更新、または解析されるオブジェクトはテキスト、URL、ドキュメント、あるいはBLOBなどです。4DのXMLオブジェクトを開くために使用されるDOMコマンドはDOM Parse XML source と DOM Parse XML variableです。

その後、要素や属性を読み込み、解析、そして書き込むために多くのコマンドを使用できます。
エラーはXML GET ERROR コマンドを使用して復旧できます (両XML標準で共通です)。

DOM CLOSE XML コマンドを使用して最後にソースを閉じます。

XML BLOB引数の使用に関する注意: XML構造はテキスト型のデータに基づいているので、XMLを扱う際にはテキスト型のフィールドや変数の利用をお勧めします。過去の経緯により、(例えばDOM Parse XML variable等) 4DのXMLコマンドはBLOB型の引数を受け入れます。以前のバージョンの4Dではテキスト型の変数が32KBに制限されていました。4Dバージョン11より、テキストフィールドおよび変数は最大2GBまでのデータを含めることができます。以前の制限は取り払われたので、今後はBLOBをテキストに格納することは全くお勧めできません。BLOBの利用はバイナリデータの処理に予約されます。XML仕様に基づき、たとえBLOBにテキストが含まれている場合でも、4D v12からはバイナリデータを自動でBase64にエンコードします。

XPath記法の利用 (DOM)

3つのXML DOMコマンド (DOM Create XML element, DOM Find XML element そして DOM SET XML ELEMENT VALUE) はXML要素にアクセスするためにXPath記法を受け入れます。
XPath記法はXPathランゲージ由来であり、XMLストラクチャ間をナビゲートする目的で使用されます。パス名タイプのシンタックスを使用して、XMLストラクチャ内で直接要素を指定できます。例えば以下の構造があるとき:

```
<RootElement>
   <Elem1>
      <Elem2>
         <Elem3 Font=Verdana Size=10> </Elem3>
      </Elem2>
   </Elem1>
</RootElement>
```

XPath記法ではElem3に /RootElement/Elem1/Elem2/Elem3 シンタックスでアクセスできます。

4DではElement[ElementNum]シンタックスを使用した**添字**によるXPath要素へのアクセスも使用できます。例えば以下の構造があるとき:

```
<RootElement>
   <Elem1>
      <Elem2>aaa</Elem2>
      <Elem2>bbb</Elem2>
      <Elem2>ccc</Elem2>
   </Elem1>
</RootElement>
```

XPath記法では /RootElement/Elem1/Elem2[3] シンタックスを使用して"ccc"値にアクセスできます。

XPath記法に関しては、DOM Create XML element と DOM Find XML element コマンドの例題を参照してください。

文字セット

以下の文字セットが4DのXML DOMとXML SAXコマンドでサポートされています:

- ASCII
- UTF-8
- UTF-16 (ビッグ/スモールエンディアン)
- UCS4 (ビッグ/スモールエンディアン)
- EBCDIC コードページ IBM037, IBM1047, IBM1140 エンコーディング,
- ISO-8859-1 (または Latin1)
- Windows-1252.

用語

XMLランゲージでは多くの特別な用語や略語が使用されます。ここでは4Dのコマンドで使用される主なXMLのコンセプトを説明します。

属性 (Attribute): 要素に付随するXMLのサブタグ。属性は常に名前と値を含みます (後述の図参照)。

子 (Child): XML構造において、ある要素の直接下のレベルにある要素。

DTD: Document Type Declaration。DTDはXMLが従わなくてはならない特定のルールとプロパティのセットを記述したものです。これらのルールは特にそれぞれのタグの名前と内容、およびそのコンテキストを定義します。この要素の形式化を使用して、XMLドキュメントが妥当であるかを検証できます。
DTDはXMLドキュメントに含めることができますし (内部DTD)、あるいは分離したドキュメントに置くこともできます (外部DTD)。DTDは必須でないことに留意してください。

要素 (Element): XMLタグ。要素は常に名前と値を含みます。オプションで要素は属性を含みます (図参照)。

要素参照 (ElementRef): 4DのXMLコマンドで使用されるXML参照はXML構造を指定します。この参照は16進化されたコードです。お使いのシステムが32 bitか64 bitかによって、16文字また32文字長となります。XML参照を格納する変数はC_TEXTディレクティブを使用して宣言します。

親 (Parent): XML構造において、ある要素の直接上のレベルにある要素。

解析、パーサ (Parsing, parser): 利用可能な情報を取り出すために構造化されたオブジェクトを解析すること。"XML"テーマのコマンドを使用してXMLオブジェクトの解析を行います。

ルート (Root): XML構造の先頭レベルに位置する要素。

兄弟 (Sibling): XML構造中、ある要素と同じレベルにある要素。

XML構造 (XML Structure): 構造化されたXMLオブジェクト。ドキュメント、変数、あるいは要素がこのオブジェクトになりえます。

妥当 (Valid): パーサにより整形式かつDTD定義に準拠していると検証されたXML。整形式を参照。

整形式 (Well-formed): パーサにより一般的なXML仕様に準拠していると検証されたXML。妥当を参照。

XML: eXtensible Markup Language。データやその構造の転送を可能にする電子的なデータ交換の標準。XMLランゲージは、HTMLのようにタグや特定のシンタックスの使用 に基づきます。しかしXMLランゲージではカスタマイズされたタグの定義が可能です。

XSL: eXtensible Stylesheet Language。XMLドキュメントへの処理および内容の表示に使用するスタイルシートの定義を行うランゲージ。

システム変数およびセット

コマンドが正しく実行されるとOKシステム変数に1が設定されます。要素参照が無効などエラーが発生した場合はOK変数に0が設定され、エラーが生成されます。この場合コマンドは空の文字列を返します。

DOM Append XML child node

DOM Append XML child node (elementRef ; childType ; childValue) -> 戻り値

引数	型		説明
elementRef	テキスト	→	XML要素参照
childType	倍長整数	→	追加する子のタイプ
childValue	テキスト, BLOB	→	子ノードとして挿入するテキストまたは (テキストあるいはBlob) 変数
戻り値	テキスト	↩	子XML要素参照

説明

DOM Append XML child node コマンドを使用して、elementRefで指定したXMLノードにchildValueの値を追加できます。

作成されるノードのタイプをchildTypeで指定します。この引数には""テーマの以下の定数を渡すことができます:

定数	型	値
XML CDATA	倍長整数	7
XML Comment	倍長整数	2
XML DATA	倍長整数	6
XML DOCTYPE	倍長整数	10
XML ELEMENT	倍長整数	11
XML Processing Instruction	倍長整数	3

childValueには挿入するデータを渡します。文字列または4D変数 (文字またはBLOB) を渡します。この引数の内容は常にテキストに変換されます。

Note: elementRef がドキュメントノード (トップレベルノード) を指す場合、コマンドは他のノードの前に"Doctype"ノードを挿入します。同じことが処理命令やコメントにも言えます。これらは常にルートノードの前 (かつDoctypeノードの後) に挿入されます。

例題 1

テキストタイプのノードを追加します:

```
Reference:=DOM Create XML element(elementRef;"myElement")
DOM SET XML ELEMENT VALUE(Reference;"Hello")
temp:=DOM Create XML element(Reference;"br")
temp:=DOM Append XML child node(Reference;XML DATA;"New")
temp:=DOM Create XML element(Reference;"br")
temp:=DOM Append XML child node(Reference;XML DATA;"York")
```

結果:

```
<myElement>Hello<br/>New<br/>York</myElement>
```

例題 2

処理命令タイプのノードを追加します:

```
$Txt_instruction:="xml-stylesheet type = \"text/xsl\" href=\"style.xsl\""
Reference:=DOM Append XML child node(elementRef;XML Processing Instruction;$Txt_instruction)
```

結果 (最初の要素の前に挿入される):

```
<?xml-stylesheet type="text/xsl"  href="style.xsl"?>
```

例題 3

コメントタイプのノードを追加する:

```
Reference:=DOM Append XML child node(elementRef;XML Comment;"Hello world")
```

結果:

```
<!--Hello  world-->
```

例題 4

CDATAタイプのノードを追加する:

```
Reference:=DOM Append XML child node(elementRef;XML CDATA;"12 < 18")
```

結果:

```
<element><![CDATA[12  < 18]]></element>
```

例題 5

Doctype 線減退のノードを追加あるいは置き換える:

```
Reference:=DOM Append XML child node(elementRef;XML DOCTYPE;"Books SYSTEM \"Book.DTD\"")
```

結果 (最初の要素の前に挿入される):

```
<!DOCTYPE  Books SYSTEM  "Book.DTD">
```

例題 6

要素タイプのノードを追加あるいは置き換える。

- childValue 引数がXMLフラグメントの場合、子ノードとして挿入されます:
  ```
  Reference:=DOM Append XML child node(elementRef;XML ELEMENT;"<child>simon</child><child>eva</child>")
  ```

  結果:

  ```
  <parent>
       <child>simon</child>
      <child>eva</child>
  </parent>
  ```
- それ以外の場合、新しい空の子要素が追加されます:
  ```
  Reference:=DOM Append XML child node(elementRef;XML ELEMENT;"break")
  ```

  Result:

  ```
  <parent>
       <break/>
   </parent>
  ```

childValue の内容が有効でない場合、エラーが返されます。

DOM Append XML element

DOM Append XML element (targetElementRef ; sourceElementRef) -> 戻り値

引数	型		説明
targetElementRef	テキスト	→	XML親要素の参照
sourceElementRef	テキスト	→	追加するXML要素の参照
戻り値	テキスト	↩	新しいXML要素参照

説明

DOM Append XML element コマンドはtargetElementRef 引数に渡した参照を持つXML要素の子要素を新しいXML要素に追加します。

sourceElementRef 引数には追加する要素の参照を渡します。この要素はDOMツリー上に既に存在するXML要素の参照として渡さなければなりません。これはtargetElementRef の子要中最後の既存の要素の後に追加されます。

例題

DOM Insert XML element コマンドの例題を参照してください。

DOM CLOSE XML

DOM CLOSE XML (elementRef)

引数	型	説明
elementRef	文字	⇒ XMLルート要素参照

説明

DOM CLOSE XML コマンドはelementRefで指定されたXMLオブジェクトにより使用されているメモリを開放します。

elementRefがXMLルートオブジェクトでない場合、エラーが生成されます。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM Count XML attributes

DOM Count XML attributes (elementRef) -> 戻り値

引数	型		説明
elementRef	文字	→	XML要素参照
戻り値	倍長整数	↩	属性数

説明

DOM Count XML attributes コマンドは、elementRefで指定したXML要素中に現れるXML属性数を返します。XML属性に関する詳細はの節を参照してください。

例題

要素の属性値を配列に受け取る前に、XML要素の属性数を取得します:

```
  <?xml version="1.0" ?>
- <LINES>
    <LINE N="1" Font="Verdana" size="10">this is a line</LINE>
    <LINE N="2" Font="charcoal" size="12">and another line</LINE>
    <LINE N="3" Font="Verdana" size="18">and we stop here</LINE>
  </LINES>
```

```
C_BLOB(myBlobVar)
C_TEXT($xml_Parent_Ref;$xml_Child_Ref)
C_TEXT(myResult)
C_LONGINT($numAttributes)

$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get first child XML element($xml_Parent_Ref)

$numAttributes:=DOM Count XML attributes($xml_Child_Ref)
ARRAY TEXT(tAttrib;$numAttributes)
ARRAY TEXT(tValAttrib;$numAttributes)
For($i;1;$numAttributes)
   DOM GET XML ATTRIBUTE BY INDEX($xml_Child_Ref;$i;tAttrib{$i};tValAttrib{$i})
End for
```

上の例題で、$numAttributesは3になり、tAttrib{1}は"Font"、tAttrib{2}は"N"、tAttrib{3}は"size"、そしてtValAttribは"Verdana","1","10"になります。

注: 配列のインデックス番号はXMLファイル中に表示される属性の順番通りではありません。XML中、属性のインデックスはnameのアルファベット順に並びかえられた属性の位置を示します。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM Count XML elements

DOM Count XML elements (elementRef ; elementName) -> 戻り値

引数	型		説明
elementRef	文字	→	XML要素参照
elementName	文字	→	数えるXML要素名
戻り値	倍長整数	↩	要素数

説明

DOM Count XML elements コマンドはelementRef で参照されるXML要素中、要素名がelementNameである子要素の数を返します。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM Create XML element

DOM Create XML element (elementRef ; xPath {; attrName ; attrValue} {; attrName2 ; attrValue2 ; ... ; attrNameN ; attrValueN}) -> 戻り値

引数	型		説明
elementRef	文字	→	ルートXML要素参照
xPath	テキスト	→	作成するXML要素のXPathパス
attrName	文字	→	設定する属性
attrValue	文字, ブール, 倍長整数, 実数, 時間, 日付	→	新しい属性値
戻り値	文字	↩	作成されたXML要素の参照

説明

DOM Create XML element コマンドは、elementRef で参照されるXML中のxPath 引数で指定された位置に新しい要素を作成し、また必要であれば属性を追加します。

elementRef にはルートの要素参照 (例えばDOM Create XML Ref コマンドで作成された) を渡します。

xPath には作成する要素のパスをXPath記法で渡します (の節の"XPath記法の利用"を参照)。途中存在しないパス要素があれば作成されます。
xPath 引数に直接単純な項目名を渡すことができます。この場合、カレントの項目のサブ項目が作成されます (例題3参照)。

Note: elementRef を使用して設定したツリーに1つ以上の名前空間が定義されている場合 (DOM Create XML Ref コマンド参照)、xPath 引数の前に名前空間を指定しなければなりません (例えば "MyNameSpace:MyElement")。

オプションのattrName と attrValue引数に必要なだけ、属性/属性値のペアを渡すことができます (変数、フィールド、またはリテラル値で)。
attrValue 引数はテキストあるいは他の型 (ブール、整数、実数、日付または時間) です。テキスト以外の型を渡した場合、4Dは以下の原則に基づきテキストへの変換を行います:

型	変換された値の例
ブール	"true" または "false"
整数	"123456"
実数	"12.34" (小数点は常に ".")
日付	"2006-12-04T00:00:00Z" (RFC 3339 標準)
時間	"5233" (秒数)

コマンドは作成された要素のXML参照を返します。

例題 1

以下の要素を作成したいとします:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<RootElement>
   <Elem1>
      <Elem2>
         <Elem3> </Elem3>
         <Elem3> </Elem3>
      </Elem2>
   </Elem1>
</RootElement>
```

これを行うには以下のコードを実行します:

```
C_TEXT(vRootRef;vElemRef)
vRootRef:=DOM Create XML Ref("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vElemRef:=DOM Create XML element(vRootRef;vxPath)
vxPath:="/RootElement/Elem1/Elem2/Elem3[2]"
vElemRef:=DOM Create XML element(vRootRef;vxPath)
```

例題 2

以下の要素を作成したいとします (属性付き):

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<RootElement>
   <Elem1>
      <Elem2>
         <Elem3 Font=Verdana Size=10> </Elem3>
      </Elem2>
   </Elem1>
</RootElement>
```

これを行うには以下のコードを実行します:

```
C_TEXT(vRootRef;vElemRef)
C_TEXT($aAttrName1;$aAttrName2;$aAttrVal1;$aAttrVal2)
$aAttrName1:="Font"
$aAttrName2:="Size"
$aAttrVal1:="Verdana"
$aAttrVal2:="10"

vRootRef:=DOM Create XML Ref("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vElemRef:=DOM Create XML element(vRootRef;vxPath;$aAttrName1;$aAttrVal1;$aAttrName2;$aAttrVal2)
```

例題 3

以下の構造を作成して書き出したいとします:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Root>
   <Elem1>Hello</Elem1>
</Root>
```

項目名を使用したシンプルなシンタックスを使用する場合、以下のように書きます:

```
C_TEXT($root)
C_TEXT($ref1)

$root:=DOM Create XML Ref("Root")
$ref1:=DOM Create XML element($root;"Elem1")
DOM SET XML ELEMENT VALUE($ref1;"Hello")
DOM EXPORT TO FILE($root;"mydoc.xml")
DOM CLOSE XML($root)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されエラーが生成されます。

エラー管理

エラーは以下の場合に生成されます:

- ルート要素参照が無効の場合。
- 作成する要素の名前が無効の場合 (例えば名前が数字で始まる場合)。

DOM Create XML element arrays

DOM Create XML element arrays (elementRef ; xPath {; attribNamesArray ; attribValuesArray} {; attribNamesArray2 ; attribValuesArray2 ; ... ; attribNamesArrayN ; attribValuesArrayN}) -> 戻り値

引数	型		説明
elementRef	テキスト	⇨	XMLルート要素参照
xPath	テキスト	⇨	作成するXML要素のXPathパス
attribNamesArray	文字配列	⇨	属性名配列
attribValuesArray	文字配列	⇨	属性値配列
戻り値	テキスト	↩	作成されたXML要素の参照

説明

DOM Create XML element arrays コマンドを使用してelementRef要素に新しい要素を追加したり、さらに配列形式で渡された属性とその値も追加できます。

配列をサポートしている以外、このコマンドはDOM Create XML elementと同じです。動作についてはこのコマンドの説明を参照してください。

さらに、DOM Create XML element arraysコマンドはattribNamesArrayとattribValuesArray引数に複数の属性とその値のペアを配列として渡すことができます。attribValuesArrayにはテキスト、日付、数値、そしてピクチャ型の配列を渡せます。4Dは自動で必要な変換を行います。新しいXML SET OPTIONSコマンドを使用してこの変換をコントロールできます。

配列は事前に作成されていなければなあらず、またペアで動作します。必要なだけ配列のペアを渡すことができ、またそれぞれのペアごとに必要なだけ要素を渡すことができます。

例題

以下の要素を作成します:

```
<?xml  version="1.0" encoding="UTF-8" standalone="no" ?>
<RootElement>
    <Elem1>
      <Elem2>
         <Elem3  Font="Verdana" Size="10" Style="Bold"></Elem3>
       </Elem2>
   </Elem1>
</RootElement>
```

これを行うには、以下のように記述します:

```
 ARRAY TEXT(arrAttNames;3)
 ARRAY TEXT(arrAttValues;3)
 arrAttNames{1}:="Font"
 arrAttValues{1}:="Verdana"
 arrAttNames{2}:="Style"
 arrAttValues{2}:="10"
 arrAttNames{3}:="Style"
 arrAttValues{3}:="Bold"
 vRootRef:=DOM Create XML Ref("RootElement")
 vxPath:="/RootElement/Elem1/Elem2/Elem3"
 vElementRef:=DOM Create XML element arrays(vRootRef;vxPath;arrAttNames;arrAttValues)
```

DOM Create XML Ref

DOM Create XML Ref (root {; nameSpace} {; nameSpaceName ; nameSpaceValue} {; nameSpaceName2 ; nameSpaceValue2 ; ... ; nameSpaceNameN ; nameSpaceValueN}) -> 戻り値

引数	型		説明
root	文字	→	ルート要素名
nameSpace	文字	→	名前空間の値
nameSpaceName	文字	→	名前空間名
nameSpaceValue	文字	→	名前空間値
戻り値	文字	⮐	ルートXML要素参照

説明

DOM Create XML Ref コマンドは空のXMLツリーをメモリに作成し、その参照を返します。

root 引数にはXMLツリーのルート要素名を渡します。

オプションのnameSpace 引数にはツリーの名前空間値の定義を渡します (例えば"http://www.4d.com")。
root引数に名前空間名とコロン、そしてルート要素名を結合した文字列を渡せることができます (例えば"MyNameSpace:MyRoot")。この場合、名前空間を指定するnameSpace引数は必須となります。

Note: 名前空間は文字列で、XML変数名がユニークであることを保証するために使用されます。一般的に、http://www.mysite.com/myurlのようなURLが使用されます。URLが有効である必要はありませんが、ユニークでなければなりません。

nameSpaceName/nameSpaceValue のペアを使用して、生成されたXMLツリーの中で1つ以上の追加の名前空間を定義できます。

重要: XMLツリーへの作業が終了したら、メモリを解放するために、DOM CLOSE XML コマンドを呼び出してください。

例題 1

ひとつのXMLツリーを作成します:

```
 C_TEXT(vElemRef)
 vElemRef:=DOM Create XML Ref("MyRoot")
```

このコードは以下の結果を生成します:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<MyRoot/>
```

例題 2

 1つの名前空間をもつXMLツリーを作成します:

```
 C_TEXT(vElemRef)
 $Root:="MyNameSpace:MyRoot"
 $Namespace:="http://www.4D.com/tech/namespace"
 vElemRef:=DOM Create XML Ref($Root;$Namespace)
```

このコードは以下の結果を生成します:

```
<MyNameSpace:MyRoot xmlns:MyNameSpace="http://www.4D.com/tech/namespace"/>
```

例題 3

複数の名前空間を持つXMLツリーを作成します:

```
 C_TEXT(vElemRef)
 C_TEXT($aNSName1;$aNSName2;$aNSValue1;$aNSValue2)
 $Root:="MyNameSpace:MyRoot"
 $Namespace:="http://www.4D.com/tech/namespace"
 $aNSName1:="NSName1"
 $aNSName2:="NSName2"
 $aNSValue1:="http://www.4D.com/Prod/namespace"
 $aNSValue2:="http://www.4D.com/Mkt/namespace"
 vElemRef:=DOM Create XML Ref($Root;$Namespace;$aNSName1;$aNSValue1;$aNSName2;$aNSValue2)
```

このコードは以下の結果を生成します:

```
<MyNameSpace:MyRoot xmlns:MyNameSpace="http://www.4D.com/tech/nameSpace"
NSName1="http://www.4D.com/Prod/namespace"
NSName2="http://www.4D.com/Mkt/namespace"/>
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

DOM EXPORT TO FILE

DOM EXPORT TO FILE (elementRef ; filePath)

引数	型	説明
elementRef	文字	→ ルートXML要素参照
filePath	テキスト	→ ファイルへのフルパス

説明

DOM EXPORT TO FILE コマンドは、XMLツリーをディスク上のファイルに格納します。

elementRef 引数には書き出すXMLのルート要素参照を渡します。

filePathには使用する、または作成するファイルのフルパスを渡します。ファイルが存在しない場合は作成されます。

ファイル名のみ (アクセスパスなし) を渡した場合、ストラクチャファイルと同階層でファイルを検索し、または作成します。

空の文字列 ("") を渡すと、標準のファイルを作成・開くダイアログが表示されます。

行末文字の処理に関する注記

XMLにおいて、改行はそれがXML要素の内側あるいは間いずれにあるかにかかわらず、大きな意味を持ちません。内部的にXMLは標準のLF文字を行区切り文字として使用します。

読み込みや書き出し処理の間、行区切り文字は変換されることがあります。読み込み時、XMLパーサーはCRLF文字 (Windowsの標準改行) をLF文字に置き換えます。書き出し時、LF文字はCR文字に置き換えられます。

改行を保持したい場合、その部分をXML CDATAセクションに含めます。そうすることによりXMLパーサーはその部分を解析対象としなくなります。CRLF文字の代わりに"<br/>"を使用することもできます。これは明示的な改行文字で、パーサーによる解析は行われません。

例題

以下の例題ではXMLツリーvElemRefをファイルMyDoc.xmlに格納します:

```
DOM EXPORT TO FILE(vElemRef;"C:\\folder\MyDoc.xml")
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

エラー管理

エラーは以下の場合に生成されます:

- ルート要素参照が無効の場合。
- アクセスパスが無効の場合。
- ボリュームがエラーを返す場合 (空き容量がない場合など)。

DOM EXPORT TO VAR

DOM EXPORT TO VAR (elementRef ; vXmlVar)

引数	型		説明
elementRef	文字	→	ルートXML要素参照
vXmlVar	テキスト, BLOB	→	XMLツリーを受け取る変数

説明

DOM EXPORT TO VAR コマンドはテキストまたはBLOB変数にXMLツリーを格納します。

elementRefには書き出すXMLのルート要素参照を渡します。

vXmlVarにはXMLツリーを受け取る変数を渡します。この変数はテキストまたはBLOBタイプでなければなりません。次に何を行うか、あるいはツリーのサイズに従ってタイプを決定できます(非Unicodeモードの場合、テキスト型の変数は32Kに制限されます。Unicodeモードの場合は2GBです)。

非Unicodeモードのとき、XMLツリーを格納するためにテキスト変数を使用すると、ツリーはカレントのMac文字セットを使用してエンコードされます (例えばほとんどのWesternシステムではMac Roman)。つまり返されるテキストは元のエンコーディング (encoding="xxx") を失います。この場合、vVarXml 変数はコードを見たり保存したりするために使用できますが、(例えばSEND PACKET コマンドを使用して書き出した場合)、有効なXMLドキュメントとはなりません。

Unicodeモードの場合、元のエンコーディングが変数内で保持されます。

行末文字の処理に関する注記

XMLにおいて、改行はそれがXML要素の内側あるいは間いずれにあるかにかかわらず、大きな意味を持ちません。内部的にXMLは標準のLF文字を行区切り文字として使用します。

読み込みや書き出し処理の間、行区切り文字は変換されることがあります。読み込み時、XMLパーサーはCRLF文字 (Windowsの標準改行) をLF文字に置き換えます。書き出し時、LF文字はCR文字に置き換えられます。

改行を保持したい場合、その部分をXML CDATAセクションに含めます。そうすることによりXMLパーサーはその部分を解析対象としなくなります。CRLF文字の代わりに"<br/>"を使用することもできます。これは明示的な改行文字で、パーサーによる解析は行われません。

例題

この例題ではXMLツリーvElemRefを変数に格納します:

```
C_TEXT(vtMyText)
DOM EXPORT TO VAR(vElemRef;vtMyText)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます (要素参照が無効な場合など)。

DOM Find XML element

DOM Find XML element (elementRef ; xPath {; arrElementRefs}) -> 戻り値

引数	型		説明
elementRef	文字	→	XML要素参照
xPath	テキスト	→	検索する要素のXPathパス
arrElementRefs	文字配列	←	見つかった要素参照のリスト (該当する場合)
戻り値	文字	↩	見つかった要素の参照 (該当する場合)

説明

DOM Find XML element コマンドはXMLストラクチャ中で特定のXML要素を検索するために使用します。検索はelementRef 引数で指定された要素から開始されます。

探すXMLノードはXPath記法を使用で xPath 引数に指定します (XML DOMコマンドの概要の"XPath記法の利用"を参照)。インデックス付きの記法を使用できます。

Note: XML標準に従い、検索は大文字小文字を区別します。

コマンドは見つけた要素のXML酸所を返します。

arrElementRefs 文字列配列が渡されると、コマンドはこの配列に見つけた要素のリストを返します。この場合コマンドは結果としてarrElementRefs 配列の最初の要素を返します。この引数はxPath 引数で指定した場所に同じ名前の要素が複数ある場合に利用できます。

例題 1

この例題は素早くXML要素を検索し、値を表示します:

```
vFound:=DOM Find XML element(vElemRef;"Items/Book[15]/Title")
DOM GET XML ELEMENT VALUE(vFound;value)
ALERT("The value of the element is: \""+value+"\"")
```

同じ検索を以下の方法で行うこともできます:

```
vFound:=DOM Find XML element(vElemRef;"Items/Book[15]")
vFound:=DOM Find XML element(vFound;"Book/Title")
DOM GET XML ELEMENT VALUE(vFound;value)
ALERT("The value of the element is: \""+value+"\"")
```

Note: 上の例題で示す通り、XPathパスは常にカレント要素の名前から始まります。これは相対XPathパスを扱う場合は特に重要です。

例題 2

以下のXML構造があるとき:

```
<Root>
   <Elem1>
      <Elem2>aaa</Elem2>
      <Elem2>bbb</Elem2>
      <Elem2>ccc</Elem2>
   </Elem1>
</Root>
```

以下のコードを使用してそれぞれのElem2要素への参照をarrAfound 配列に受け取ります:

```
ARRAY TEXT(arrAfound;0)
vFound:=DOM Find XML element(vElemRef;"/Root/Elem1/Elem2";arrAfound)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

エラー管理

エラーは以下の場合に生成されます:

- 要素参照が無効の場合。
- 指定されたXPathパスが無効の場合。

DOM Find XML element by ID

DOM Find XML element by ID (elementRef ; id) -> 戻り値

引数	型		説明
elementRef	文字	→	XML要素参照
id	文字	→	検索する要素のID属性値
戻り値	文字	↩	見つけた要素の参照 (該当する場合)

説明

DOM Find XML element by ID コマンドはXMLドキュメント中で、id属性値が引数として渡したidと同じである要素を検索します。

elementRefにはXMLドキュメント中の要素の参照を渡します。ルート要素の参照あるいは他の要素の参照を渡すことができます。検索はelementRefの位置を考慮せず、常にドキュメント全体を検索します。

コマンドは見つけたXML要素の参照を返します。

警告: XMLにおいて、id属性はドキュメント要素ごとのユニークIDを関連付けます。ID属性の値は有効なXML名でなければならず、XMLドキュメント内のすべての要素中でユニークでなければなりません (妥当性制約)。DOM Find XML element by IDコマンドが正しく動作するためには、この制約が守られていなければなりません。そうでない場合、結果は保証されません (コマンドはドキュメント中で最初に見つけた要素を返します)。

DOM Get first child XML element

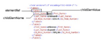
DOM Get first child XML element (elementRef {; childElemName {; childElemValue}}) -> 戻り値

引数	型		説明
elementRef	文字	→	XML要素参照
childElemName	文字	←	子要素名
childElemValue	文字	←	子要素値
戻り値	文字	↩	子要素参照 (16 文字)

説明

DOM Get first child XML element コマンドはelementRefに渡した要素の最初の子要素への参照を返します。この参照は他のXML解析コマンドで使用できます。

childElemName と childElemValue 引数が渡されると、子要素の名前と値がそれぞれ返されます。



例題 1

ルートの最初の子要素の参照を取得します。XML構造 (C:\\import.xml) はまずBLOBにロードされます:

```
C_BLOB(myBlobVar)
C_TEXT($xml_Parent_Ref;$xml_Child_Ref)

DOCUMENT TO BLOB("c:\\import.xml";myBlobVar)
$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get first child XML element($xml_Parent_Ref)
```

例題 2

ルートの最初の子要素の参照、名前および値を取得します。XML構造 (C:\\import.xml) はまずBLOBにロードされます:

```
C_BLOB(myBlobVar)
C_TEXT($xml_Parent_Ref;$xml_Child_Ref)
C_TEXT($childName;$childValue)

DOCUMENT TO BLOB("c:\\import.xml";myBlobVar)
$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get first child XML element($xml_Parent_Ref;$childName;$childValue)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM Get last child XML element

DOM Get last child XML element (elementRef {; childElemName {; childElemValue}}) -> 戻り値

引数	型		説明
elementRef	文字	⇒	XML要素参照
childElemName	文字	⇐	子要素名
childElemValue	文字	⇐	子要素値
戻り値	文字	⤷	XML要素参照 (16 文字)

説明

DOM Get last child XML element コマンドはelementRefに渡した要素の最後の子要素への参照を返します。この参照は他のXML解析コマンドで使用できます。

childElemName と childElemValue 引数が渡されると、子要素の名前と値がそれぞれ返されます。

例題

ルートの最後の子要素の参照を取得します。XML構造 (C:\\import.xml) はまずBLOBにロードされます:

```
C_BLOB(myBlobVar)
C_TEXT($ref_XML_Parent;$ref_XML_Child)
C_TEXT($childName;$childValue)

DOCUMENT TO BLOB("c:\\import.xml";myBlobVar)
$ref_XML_Parent:=DOM Parse XML variable(myBlobVar)
$ref_XML_Child:=DOM Get last child XML element($ref_XML_Parent;$childName;$childValue)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM Get next sibling XML element

DOM Get next sibling XML element (elementRef {; siblingElemName {; siblingElemValue}}) -> 戻り値

引数	型		説明
elementRef	文字	→	XML要素参照
siblingElemName	文字	←	兄弟XML要素名
siblingElemValue	文字	←	兄弟XML要素値
戻り値	文字	↩	兄弟XML要素参照 (16 文字)

説明

DOM Get next sibling XML element コマンドは参照として渡したXML要素の次の兄弟要素の参照を返します。この参照は他のXML解析コマンドで使用できます。

siblingElemName と siblingElemValue 引数が渡されると、兄弟要素の名前と値がそれぞれ返されます。

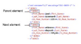
このコマンドはXMLの子要素の間をナビゲートするために使用されます。

最後の兄弟要素の後、システム変数は0に設定されます。

例題 1

引数として渡した要素の次の兄弟要素の参照を取得します:

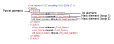
```
C_TEXT($xml_Parent_Ref;$next_XML_Ref)
$next_XML_Ref:=DOM Get next sibling XML element($xml_Parent_Ref)
```



例題 2

引数で渡した親要素のすべての子要素をループで参照します:

```
C_TEXT($xml_Parent_Ref;$first_XML_Ref;$next_XML_Ref)

$first_XML_Ref:=DOM Get first child XML element($xml_Parent_Ref)
$next_XML_Ref:=$first_XML_Ref
While(OK=1)
   $next_XML_Ref:=DOM Get next sibling XML element($next_XML_Ref)
End while
```



システム変数およびセット

コマンドが正しく実行され、解析された要素が参照された要素の最後の兄弟要素でない場合、システム変数OKに1が設定されます。エラーが発生したり、解析された要素が参照された要素の最後の兄弟要素である場合、0が設定されます。

DOM Get parent XML element

DOM Get parent XML element (elementRef {; parentElemName {; parentElemValue}}) -> 戻り値

引数	型		説明
elementRef	文字	⇒	XML要素参照
parentElemName	文字	⇐	親XML要素名
parentElemValue	文字	⇐	親XML要素値
戻り値	文字	↩	親XML要素参照 (16 文字)

説明

DOM Get parent XML elementコマンドは、elementRefに参照で渡したXML要素の親要素参照を返します。この参照は他のXML解析コマンドで使用できます。

オプションのparentElemName と parentElemValue引数が渡されると、それぞれ親要素の名前と値を受け取ります。

elementRefにルート要素を渡した場合、コマンドは"#document"参照を返します。このdocumentノードはルート要素の親です。
documentノードにこのコマンドを適用すると、コマンドはヌル参照 ("0000000000000000") を返し、OKシステム変数が0に設定されます。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM Get previous sibling XML element

DOM Get previous sibling XML element (elementRef {; siblingElemName {; siblingElemValue}}) -> 戻り値

引数	型		説明
elementRef	文字	→	XML要素参照
siblingElemName	文字	←	兄弟XML要素名
siblingElemValue	文字	←	兄弟XML要素値
戻り値	文字	↩	兄弟XML要素参照 (16 文字)

説明

DOM Get previous sibling XML element コマンドは参照として渡したXML要素の前の兄弟要素の参照を返します。この参照は他のXML解析コマンドで使用できます。

siblingElemName と siblingElemValue 引数が渡されると、前の兄弟要素の名前と値がそれぞれ返されます。

このコマンドはXMLの子要素の間をナビゲートするために使用されます。

最初の兄弟要素の前、システム変数は0に設定されます。

システム変数およびセット

コマンドが正しく実行され、解析された要素が参照された要素の最初の子要素でない場合、システム変数OKに1が設定されます。エラーが発生したり、解析された要素が参照された要素の最初の子要素である場合、0が設定されます。

DOM Get Root XML element

DOM Get Root XML element (elementRef) -> 戻り値

引数	型		説明
elementRef	文字	→	XML要素参照
戻り値	文字	↩	ルート要素参照 (16 文字) またはエラーの場合 ""

説明

DOM Get Root XML element コマンドはelementRef 引数に渡したXML要素が属するXMLドキュメントのルート要素参照を返します。この参照は他のXML解析コマンドで使用できます。

DOM GET XML ATTRIBUTE BY INDEX

DOM GET XML ATTRIBUTE BY INDEX (elementRef ; attribIndex ; attribName ; attribValue)

引数	型		説明
elementRef	文字	⇨	XML要素参照
attribIndex	倍長整数	⇨	属性のインデックス番号
attribName	変数	⇦	属性名
attribValue	変数	⇦	属性値

説明

DOM GET XML ATTRIBUTE BY INDEX コマンドは、属性のインデックスを使用して、属性名と値を取得するために使用します。

elementRef にはXML要素参照を、attribIndexには名前を知りたい要素のインデックス番号を渡します。名前がattribName 引数に、値がattribValue 引数に返されます。4Dは取得した値を引数として渡した変数と同じ型に変換します。

Note: 配列のインデックス番号はXMLファイル中に表示される属性の順番通りではありません。XML中、属性のインデックスはnameのアルファベット順に並びかえられた属性の位置を示します。この点についてはDOM Count XML attributes コマンドの説明を参照してください。

attribIndex に渡された値がXML要素に定義された属性数より多い場合、エラーが返されます。

例題

DOM Count XML attributes コマンドの例題参照

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM GET XML ATTRIBUTE BY NAME

DOM GET XML ATTRIBUTE BY NAME (elementRef ; attribName ; attribValue)

引数	型		説明
elementRef	文字	⇒	XML要素参照
attribName	文字	⇒	属性名
attribValue	変数	⇐	属性値

説明

DOM GET XML ATTRIBUTE BY NAME コマンドを使用して、属性名に対応する属性値を取得できます。

elementRef に要素参照を、attribNameには属性値を取得したい属性の名前を渡します。値はattribValue 引数に返されます。4Dは取得した値を渡した変数と同じ型に変換します。

attribName 属性がXML要素中に存在しない場合、エラーが返されます。複数の同じ名前を持つ属性がXML要素中に存在する場合、最初の属性のみが返されます。

例題

このメソッドは名前を指定してXML属性を取得するために使用します:

```
C_BLOB(myBlobVar)
C_TEXT($xml_Parent_Ref;$xml_Child_Ref)
C_LONGINT($LineNum)

$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get first child XML element($xml_Parent_Ref)
DOM GET XML ATTRIBUTE BY NAME($xml_Child_Ref;"N";$LineNum)
```

このメソッドを以下の例題に適用すると、$LineNumには1が返されます:

```
  <?xml version="1.0" ?>
- <STANZA>
    <LINE N="1">I heard a thousand blended notes,</LINE>
    <LINE N="2">While in grove I sate reclined,</LINE>
    <LINE N="3">In that sweet mood when pleasant thoughts</LINE>
    <LINE N="4">Bring sad thoughts to the mind.</LINE>
  </STANZA>
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM GET XML CHILD NODES

DOM GET XML CHILD NODES (elementRef ; childTypesArr ; nodeRefsArr)

引数	型		説明
elementRef	テキスト	→	XML要素参照
childTypesArr	倍長整数配列	←	子ノードのタイプ
nodeRefsArr	テキスト配列	←	子ノードの参照または値

説明

DOM GET XML CHILD NODES コマンドはelementRefで指定したXML要素のすべての子ノードのタイプと参照または値を返します。

子ノードのタイプはchildTypesArr配列に返されます。返された値は""テーマの以下の定数と比較できます:

定数	型	値
XML Comment	倍長整数	2
XML Processing Instruction	倍長整数	3
XML DATA	倍長整数	6
XML CDATA	倍長整数	7
XML DOCTYPE	倍長整数	10
XML ELEMENT	倍長整数	11

詳細はDOM Append XML child node コマンドの説明を参照してください。

nodeRefsArr 配列には、(内容または指示命令に基づき) 要素の値または参照が返されます。

例題

以下のXML構造があるとき:

```
<myElement>Hello<br/>New<br/>York</myElement>
```

以下のコードを実行後:

```
 elementRef:=DOM Find XML element($root;"myElement")
 DOM GET XML CHILD NODES(elementRef;$typeArr;$textArr)
```

$typeArrと$textArr配列には以下の値が含まれます:

$typeArr{1}=6 $textArr{1} = "Hello"
$typeArr{2}=11 $textArr{2} = "AEF1233456878977" (<br/>の要素参照)
$typeArr{3}=6 $textArr{3} = "New"
$typeArr{4}=11 $textArr{4} = "AEF1237897734568" (<br/>の要素参照)
$typeArr{5}=6 $textArr{5} = "York"

DOM Get XML document ref

DOM Get XML document ref (elementRef) -> 戻り値

引数	型		説明
elementRef	テキスト	→	DOMツリー中の既存の要素の参照
戻り値	テキスト	↩	DOMツリーの最初の要素の参照 (ドキュメントノード)

説明

DOM Get XML document ref コマンドを使用してelementRefに渡したDOMツリーの"ドキュメント"参照を取得できます。ドキュメント要素はDOMツリーの最初の要素であり、ルート要素の親です。

ドキュメント要素の参照を使用して"Doctype"や"処理命令"ノードを処理できます。これはDOM Append XML child nodeとDOM GET XML CHILD NODESコマンドでのみ利用できます。

このレベルでは、処理命令やコメントを追加したり、Doctypeノードを置換したりすることだけが可能です。ここにCDATAやテキストノードを作成することはできません。

例題

この例題では、XMLドキュメントのDTD宣言を取得します:

```
C_TEXT($rootRef)
$rootRef:=DOM Parse XML source("")
If(OK=1)
   C_TEXT($documentRef)
 // ドキュメントノードを探します。このノードにはルートノード
 // の前にDOCTYPEが記述されています。
   $documentRef:=DOM Get XML document ref($rootRef)
   ARRAY TEXT($typeArr;0)
   ARRAY TEXT($valueArr;0)
 // このノードの子ノード中でDOCTYPEタイプのノードを探す
   DOM GET XML CHILD NODES($refDocument;$typeArr;$valueArr)
   C_TEXT($text)
   $text:=""
   $pos:=Find in array($typeArr;XML DOCTYPE)
   If($pos>-1)
 // DTD宣言を$textに取得
      $text:=$text+"Doctype: "+$valueArr{$pos}+Char(Carriage return)
   End if
   DOM CLOSE XML($rootRef)
End if
```

DOM Get XML element

DOM Get XML element (elementRef ; elementName ; index ; elementValue) -> 戻り値

引数	型		説明
elementRef	文字	⇨	XML要素参照
elementName	文字	⇨	取得する要素の名前
index	倍長整数	⇨	取得する要素のインデックス番号
elementValue	変数	⇦	要素値
戻り値	文字	↩	XML参照 (16 文字)

説明

DOM Get XML element コマンドは、elementName と index 引数に基づき、子要素の参照を返します。

要素の値がelementValue 引数に返されます。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM GET XML ELEMENT NAME

DOM GET XML ELEMENT NAME (elementRef ; elementName)

引数	型		説明
elementRef	文字	→	XML要素参照
elementName	変数	←	要素の名前

説明

DOM GET XML ELEMENT NAME コマンドは、elementRefで指定したXML要素の名前をelementName 引数に返します。XML要素名に関する詳細はの節を参照してください。

例題

このメソッドは$xml_Element_Ref要素の名前を返します:

```
C_TEXT($xml_Element_Ref)
C_TEXT($name)

DOM GET XML ELEMENT NAME($xml_Element_Ref;$name)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM GET XML ELEMENT VALUE

DOM GET XML ELEMENT VALUE (elementRef ; elementValue {; cDATA})

引数	型		説明
elementRef	文字	→	XML要素参照
elementValue	変数	←	要素値
cDATA	変数	←	CDATAセクションの内容

説明

DOM GET XML ELEMENT VALUE コマンドは elementRefに指定したXML要素の値をelementValue 引数に返します。4Dは取得した値を渡した変数と同じ型に変換します。

オプションの cDATA 引数を使用して、elementRef 要素のCDATAセクションの値を所得できます。elementValue 引数のように、4Dは取得した値を渡した変数と同じ型に変換します。

Note: elementRefで指定された要素がDOM SET XML ELEMENT VALUE コマンドで処理されたBLOBの場合、それは自動でBase64でエンコードされています。このコマンドは自動でBase64のデコードを試みます。

例題

このメソッドは$xml_Element_Ref要素の値を返します:

```
C_TEXT($xml_Element_Ref)
C_REAL($value)

DOM GET XML ELEMENT VALUE($xml_Element_Ref;$value)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM Get XML information

DOM Get XML information (elementRef ; xmlInfo) -> 戻り値

引数	型		説明
elementRef	文字	→	XMLルート要素参照
xmlInfo	倍長整数	→	取得する情報のタイプ
戻り値	文字	←	XML情報の値

説明

DOM Get XML informationコマンドを使用して、elementRefで指定したXML要素に関する様々な情報を取得できます。

xmlInfoには、取得する情報のタイプを指定するコードを渡します。テーマの以下の定数を使用できます:

定数	型	値	コメント
DOCTYPE Name	倍長整数	3	DOCTYPE マーカで定義されているルート要素の名前。
Document URI	倍長整数	6	DTDのURI
Encoding	倍長整数	4	使用されているエンコーディング (UTF-8, ISO...).
PUBLIC ID	倍長整数	1	ドキュメントが従うDTDの公開識別子 (FPI, DOCTYPE xxx PUBLICタグが存在する場合)。
SYSTEM ID	倍長整数	2	システム識別子。
Version	倍長整数	5	受け入れられたXMLバージョン。

DOM Insert XML element

DOM Insert XML element (targetElementRef ; sourceElementRef ; childIndex) -> 戻り値

引数	型	説明
targetElementRef	テキスト	→ 親XML要素参照
sourceElementRef	テキスト	→ 挿入するXML要素参照
childIndex	倍長整数	→ 新しい要素を挿入するターゲットとなる子要素のインデックス
戻り値	テキスト	↩ 新しいXML要素の参照

説明

DOM Insert XML element コマンドを使用してtargetElementRef引数に渡された参照を持つXML要素の子要素の間に、新しいXML要素を挿入できます。

sourceElementRefに挿入する要素を渡します。この要素は、DOMツリーの中の既存のXML要素の参照として渡さなければなりません。

childIndex引数は、新しい要素を挿入する、親要素の子要素を指定するために使用します。この引数にはインデックス番号を渡します。番号が有効でない場合 (例えばこのインデックス番号を持つ子要素が存在しない)、新しい要素は親要素の最初の子要素の前に挿入されます。

コマンドは取得したXML要素の参照を返します。

例題

以下のXML構造で、1番目と2番目の本を入れ替えます:

```
<?xml version="1.0" encoding="UTF-8"  standalone="no" ?>
<BookCatalog>
  <Book>
           <Title>Open Source Web Services</Title>
           <Author>Collective</Author>
           <Date>2003</Date>
           <ISBN>2-7440-1507-5</ISBN>
           <Publisher>Wrox</Publisher>
    </Book>
<Book>
           <Title>Building XML Web services</Title>
           <Author>Scott Short</Author>
           <Date>2002</Date>
           <ISBN>2-10-006476-2</ISBN>
           <Publisher>Microsoft Press</Publisher>
    </Book>
</BookCatalog>
```

これを行うには、以下のコードを実行します:

```
 C_TEXT($rootRef)
 $rootRef:=DOM Parse XML source("") // XMLドキュメントを選択
 If(OK=1)
    C_TEXT($newStruct)
    $newStruct:=DOM Create XML Ref("NewBooks")&NBSP; // 新しく構造を作成

    $bookRef:=DOM Find XML element($rootRef;"/BookCatalog/Book[1]")
    $newElementRef:=DOM Append XML element($newStruct;$bookRef)

    $bookRef:=DOM Find XML element($rootRef;"/BookCatalog/Book[2]")
    C_TEXT($newElementRef)
    $newElementRef:=DOM Insert XML element($newStruct;$bookRef;1)

    DOM CLOSE XML($newStruct)
    DOM CLOSE XML($rootRef)
 End if
```

DOM Parse XML source

DOM Parse XML source (document {; validation {; dtd | schema}}) -> 戻り値

引数	型		説明
document	文字	→	ドキュメントのパス名
validation	ブール	→	True = 検証を行う False = 検証を行わない
dtd | schema	文字	→	DTDまたはXMLスキーマの場所
戻り値	文字	↩	XML要素参照 (16 文字)

説明

DOM Parse XML sourceコマンドはXML構造を含むドキュメントを解析し、XMLツリーへの参照を返します。このコマンドはDTDやXMLスキーマ (XML Schema Definition (XSD) ドキュメント) を使用して、ドキュメントを検証したり、あるいはしないこともできます。
ドキュメントはディスク上あるいはイントラネットやインターネット上に存在できます。

備考: DOM Parse XML sourceコマンドの実行は同期しています。

document 引数引数には以下を渡します:

- 標準の完全パス名 (WindowsでのC:\\Folder\\File\\...やMac OSでのMacintoshHD:Folder:File)、
- またはMsc OS上ではUnixパス名 (/で始まらなければなりません)、
- またはhttp://www.site.com/File や ftp://public.ftp.com...のようなネットワークパス。

ブール引数 validation には構造を検証するかどうかを指定します。

- validation がTrueの場合、構造は検証されます。この場合、パーサはドキュメントのXML構造を、ドキュメントに含まれるDTDまたはXSD参照、または3番目の引数で指定されたにDTDまたはXSD参照基づいて検証を試みます。
- validation がFalseの場合、構造は検証されません。
- validation にTrueを渡し、3番目の引数を省略する場合、コマンドは構造自身の中で見つけるDTDやXSD参照を使用して検証を行います。検証を間接的に行うこともできます。構造がDTDへの参照を含んでおり、それ自身にXSDファイルへの参照も含む場合、コマンドは両方の検証を試みます。

3番目の引数を使用して、ドキュメントの解析に使用する特定のDTDやXMLスキーマを指定できます。この引数を使用する場合、コマンドはXMLドキュメント内で参照されるDTDを考慮しません。

DTDによる検証

DTDを指定する方法は2つあります:

- 参照として: この方法を使用するには、dtd 引数に".dtd"拡張子を持つDTDの完全パス名を渡します。指定したドキュメントに有効なDTDが含まれていない場合、dtd 引数は無視されエラーが生成されます。
- テキストに直接: この場合、引数の内容が"<?xml"で始まっていれば、4DはそれをDTDとして扱います。そうでなければ4Dはそれをパス名として扱います。

スキーマによる検証

XMLスキーマでドキュメントを検証するには、3番目の引数に.dtd拡張子の代わりに.xsd拡張子を持つファイルやURLのパスを渡します。XMLスキーマによる検証はDTDによるそれよりも自由度が高くパワフルです。XSDドキュメントのランゲージはXMLランゲージに基づいています。特にXMLスキーマはデータタイプをサポートします。XMLスキーマに関する情報は以下のアドレスを参照してください: http://www.w3.org/XML/Schema.

DTDやXSDが存在しなかったりURLが正しくない場合など検証が行えない場合、エラーが生成されます。Errorシステム変数はエラー番号を示します。ON ERR CALL コマンドを使用してインストールされるエラー処理メソッドを使用して、このエラーをとらえることができます。

このコマンドは、メモリ中に展開されたドキュメントの仮想構造への参照を表す16-文字の文字列 (ElementRef) を返します。この参照を他のXML解析コマンドで使用できます。

重要: 参照の利用が終了したら、DOM CLOSE XML コマンドを使用してこの参照が使用しているメモリを解放することを忘れないでください。

例題 1

検証なしでディスク上のXMLドキュメントを開きます:

```
$xml_Struct_Ref:=DOM Parse XML source("C:\\import.xml")
```

例題 2

検証なしで、データベースストラクチャと同階層にあるXMLドキュメントを開きます:

```
$xml_Struct_Ref:=DOM Parse XML source("import.xml")
```

例題 3

ディスク上のDTDを使用した検証を行い、ディスク上のXMLドキュメントを開きます:

```
$xml_Struct_Ref:=DOM Parse XML source("C:\\import.xml";True;"C:\\import_dtd.xml")
```

例題 4

検証なしで特定のURLに存在するXMLドキュメントを開きます:

```
$xml_Struct_Ref:=DOM Parse XML source("http://www.4D.com/xml/import.xml")
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM Parse XML variable

DOM Parse XML variable (variable {; validation {; dtd | schema}}) -> **戻り値**

引数	型		説明
variable	BLOB, テキスト	⇒	変数名
validation	ブール	⇒	True = DTDによる検証を行う False = 検証なし
dtd | schema	文字	⇒	DTDまたはXMLスキーマの場所
戻り値	文字	↩	XML要素参照 (16 文字)

説明

DOM Parse XML variableコマンドはXML構造を格納するBLOBまたはテキスト型変数を解析し、XML構造への参照を返します。コマンドはDTDやXMLスキーマ (XML Schema Definition (XSD) ドキュメント) を使用してドキュメントの検証を行うこともできます。

variable 引数にはXMLオブジェクトを含むBLOBまたはテキスト変数の名前を渡します。

ブール引数 validation はDTDを使用して構造の検証を行うか行わないかを指定します。

- validation がTrueの場合、ストラクチャは検証されます。この場合、パーサはドキュメント中で定義または参照されるDTDもしくはXSD、または三番目の引数が渡された場合はこの引数で指定されたDTDもしくはXSDに基づき、XML構造を検証します。
- validation がFalseの場合、ストラクチャは検証されません。

validationにTrueを渡し、三番目の引数を省略すると、コマンドはXMLストラクチャ中に含まれるDTDまたはXSD参照を使用して、XMLストラクチャを検証しようとします。間接検証も可能です。XMLストラクチャにDTDへの参照が含まれていて、さらにそこにXSDファイルへの参照が含まれる場合、コマンドは両方の検証を試みます。

3番目の引数はドキュメントの解析に使用するDTDやXMLスキーマを指定するために使用します。この引数を使用すると、コマンドはXML変数内で参照されるDTDを考慮に入れません。

DTDによる検証

DTDを指定する方法は2つあります:

- 参照として: この方法を使用するには、dtd 引数に".dtd"拡張子を持つDTDの完全パス名を渡します。指定したドキュメントに有効なDTDが含まれていない場合、dtd 引数は無視されエラーが生成されます。
- テキストに直接: この場合、引数の内容が"<?xml"で始まっていれば、4DはそれをDTDとして扱います。そうでなければ4Dはそれをパス名として扱います。

スキーマによる検証

ドキュメントをXMLスキーマで検証するには、三番目の引数に"dtd"の代わりに、"xsd"拡張子を持つファイルまたはURLへのパスを渡します。XMLスキーマによる検証はDTDのそれに比べより自由度がありパワフルであるといわれています。XSDドキュメントのランゲージはXMLに基づきます。特に、XMLスキーマはデータタイプをサポートします。XMLスキーマに関する詳細は、以下のWebサイトを参照してください:
http://www.w3.org/XML/Schema

DTDやXSDが存在しなかったり、URLが正しくない場合など検証が行えない場合、エラーが生成されます。Errorシステム変数はエラー番号を示します。ON ERR CALLコマンドを使用してインストールされるエラー処理メソッドを使用して、このエラーをとらえることができます。

このコマンドは、メモリ中に展開されたドキュメントの仮想構造への参照を表す文字列 (ElementRef) を返します。この参照を他のXML解析コマンドで使用できます。

重要: 参照の利用が終了したら、DOM CLOSE XMLコマンドを使用してこの参照が使用しているメモリを解放することを忘れないでください。

例題 1

検証なしで、4Dテキスト変数に格納されたXMLを開きます:

```
C_TEXT(myTextVar)
C_TIME(vDoc)
C_TEXT($xml_Struct_Ref)

vDoc:=Open document("Document.xml")
If(OK=1)
   RECEIVE PACKET(vDoc;myTextVar;32000)
   CLOSE DOCUMENT(vDoc)
   $xml_Struct_Ref:=DOM Parse XML variable(myTextVar)
End if
```

例題 2

検証なしで、4D BLOBに格納されたXMLを開きます:

```
C_BLOB(myBlobVar)
C_TEXT($ref_XML_Struct)

DOCUMENT TO BLOB(c\\import.xml;myBlobVar)
$xml_Struct_Ref:=DOM Parse XML variable(myBlobVar)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

DOM REMOVE XML ATTRIBUTE

DOM REMOVE XML ATTRIBUTE (elementRef ; attribName)

引数	型		説明
elementRef	テキスト	⇒	XML要素参照
attribName	テキスト	⇒	取り除く属性

説明

DOM REMOVE XML ATTRIBUTE コマンドはelementRefで指定されたXML要素にattrNameで指定された属性が存在すれば、それを取り除きます。

属性が正しく取り除かれると、OKシステム変数に1が設定されます。elementRef要素にattrNameという名前の属性が存在しない場合、エラーが返され、OKシステム変数に0が設定されます。

例題

以下のXML構造において:

```
  <?xml version="1.0" ?>
- <STANZA>
    <LINE N="1">I heard a thousand blended notes,</LINE>
    <LINE N="2">While in grove I sate reclined,</LINE>
    <LINE N="3">In that sweet mood when pleasant thoughts</LINE>
    <LINE N="4">Bring sad thoughts to the mind.</LINE>
  </STANZA>
```

以下のコードは一番目の属性"N=1"を取り除きます:

```
C_BLOB(myBlobVar)
C_TEXT($xml_Parent_Ref;$xml_Child_Ref)
C_LONGINT($LineNum)

$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get first child XML element($xml_Parent_Ref)
DOM REMOVE XML ATTRIBUTE($xml_Child_Ref;"N")
```

DOM REMOVE XML ELEMENT

DOM REMOVE XML ELEMENT (elementRef)

引数	型		説明
elementRef	文字	→	XML要素参照

説明

DOM REMOVE XML ELEMENT コマンドはelementRefで指定した要素を取り除きます。

システム変数およびセット

コマンドが正しく実行されると、システム変数OKは1に設定されます。そうでなければ0に設定されエラーが生成されます。
エラーは要素参照が無効な場合に生成されます。

DOM SET XML ATTRIBUTE

DOM SET XML ATTRIBUTE (elementRef ; attrName ; attrValue {; attrName2 ; attrValue2 ; ... ; attrNameN ; attrValueN})

引数	型		説明
elementRef	文字	→	XML要素参照
attrName	文字	→	設定する属性
attrValue	文字, ブール, 倍長整数, 実数, 時間, 日付	→	新しい属性値

説明

DOM SET XML ATTRIBUTE コマンドを使用して、elementRef に渡したXML要素に1つ以上の属性を追加できます。また定義されたそれぞれの属性に値を設定することもできます。

attrName と attrValue引数にそれぞれ設定する属性とその値を (変数、フィールド、またはリテラル値の形式で) 渡します。必要なだけ属性/値のペアを渡すことができます。
attrValue 引数はテキストあるいは (ブール、整数、実数、日付または時間など) 他のタイプを渡すことができます。テキスト以外の値を渡した場合、4Dは以下の原則に基づきテキストに変換します:

型	変換された値の例
ブール	"true" または "false"
整数	"123456"
実数	"12.34" (小数点は常に ".")
日付	"2006-12-04T00:00:00Z" (RFC 3339 標準)
時間	"5233" (秒数)

例題

以下のXMLソースがあるとき:

```
<Book>
   <Title>The Best Seller</Title>
</Book>
```

以下のコードを実行すると:

```
 vAttrName:="Font"
 vAttrVal:="Verdana"
 DOM SET XML ATTRIBUTE(vElemRef;vAttrName;vAttrVal)
```

以下のようになります:

```
<Book>
   <Title Font=Verdana>The Best Seller</Title>
</Book>
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

DOM SET XML DECLARATION

DOM SET XML DECLARATION (elementRef ; encoding {; standalone {; indentation}})

引数	型		説明
elementRef	文字	→	XML要素参照
encoding	文字	→	XMLドキュメント文字セット
standalone	ブール	→	True = ドキュメントはスタンドアロン False (デフォルト) = ドキュメントはスタンドアロンではない
indentation	ブール	→	***廃止予定、使用しないでください***

説明

DOM SET XML DECLARATIONコマンドを使用して、elementRefにより設定されるXMLツリーの作成に利用されるさまざまなオプションを設定できます。これらのオプションはツリーのエンコーディングやスタンドアロンプロパティに関連します:

- encoding: ドキュメント中で使用されるエンコーディングを指定します。デフォルトで (コマンドが呼び出されないと) UTF-8文字セットが使用されます。
- standalone: ツリーがスタンドアロン (True) か、それが他のファイルや外部リソースを処理のために必要とするか (False) を指定します。デフォルトで (コマンドが呼び出されないか引数が省略されると) 、ツリーはスタンドアロンではありません。

互換性に関するメモ: indentation引数は以前のバージョンの4Dとの互換性のために保持されていますが、その利用はv12では推奨されません。今後ドキュメントのインデントを指定するには、XML SET OPTIONSコマンドの利用を強く推奨します。

例題

以下の例題はelementRef要素で使用するエンコーディングとスタンドアロンオプションを設定します:

```
DOM SET XML DECLARATION(elementRef;"UTF-16";True)
```

DOM SET XML ELEMENT NAME

DOM SET XML ELEMENT NAME (elementRef ; elementName)

引数	型		説明
elementRef	文字	⇒	XML要素参照
elementName	文字	⇒	要素の新しい名前

説明

DOM SET XML ELEMENT NAME コマンドを使用して、elementRefで指定した要素の名前を変更します。

名称を変更する要素の参照をelementRef に渡します。コマンドは要素の開く、閉じるタグの更新も行います。

例題

以下のXMLソースにおいて:

```
<Book>
   <Title>The Best Seller</Title>
</Book>
```

vElemRefが'Book'要素への参照をもつときに、以下のコードを実行すると:

```
 DOM SET XML ELEMENT NAME(vElemRef;"BestSeller")
```

以下のようになります:

```
<BestSeller>
   <Title>The Best Seller</Title>
</BestSeller>
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

エラー管理

以下のような場合にエラーが生成されます:

- 要素参照が無効である。
- 新しい要素名が無効である (例えば要素名が数字から始まっているなど)。

DOM SET XML ELEMENT VALUE

DOM SET XML ELEMENT VALUE (elementRef {; xPath}; elementValue {; *})

引数	型		説明
elementRef	文字	→	XML要素参照
xPath	テキスト	→	XML要素のXPathパス
elementValue	文字, 変数	→	要素の新しい値
*	演算子	→	指定時: 値をCDATAに設定

説明

DOM SET XML ELEMENT VALUEコマンドを使用して、elementRefで指定した要素の値を更新できます。

オプションのxPath 引数を渡すと、更新する要素の指定にXPath記法を指定したことになります (この記法に関する詳細は、XML DOMコマンドの概要の "XPath記法の利用" を参照してください)。この場合、elementRef にはルート要素参照を渡し、xPath には変更する要素のXPathを渡します。

elementValueには新しい値を格納した文字列、変数またはフィールドを渡します:

- 文字列を渡した場合、値はXML構造の中でそのまま使用されます。
- 変数またはフィールドを渡した場合、4DはelementValueの型に基づき値を処理します。配列やピクチャ、ポインタを除くすべてのデータタイプを使用できます。

オプションのアスタリスクを渡すと、要素の値はCDATAセクションに設定されます。テキストをそのまま挿入するために特別なCDATAの形式が使用されます (例題 2参照)。

注: elementRefで指定された要素が**DOM SET XML ELEMENT VALUE**コマンドで処理されたBLOBの場合、それは自動でBase64にエンコードされます。この場合DOM GET XML ELEMENT VALUEは自動で逆の処理を行います。

例題 1

以下のXMLソースで:

```
<Book>
   <Title>The Best Seller</Title>
</Book>
```

vElemRef が"Title"要素への参照を持つときに以下のコードを実行すると:

```
 DOM SET XML ELEMENT VALUE(vElemRef;"The Loser")
```

以下のようになります:

```
<Book>
   <Title>The Loser</Title>
</Book>
```

例題 2

以下のXMLソースがあるとき:

```
 <Maths>
   <Postulate>1+2=3</Postulate>
</Maths>
```

<Postulate> 要素にテキスト "12<18" を書き込みたいとします。"<" 文字は受け入れられないため、この文字列をそのままXMLに書き込むことはできません。この文字を "<" に変更するか、CDATAの形式を使用しなければなりません。vElemRefがXML <Postulate> ノードを指すとき:

```
  ` Normal form
 DOM SET XML ELEMENT VALUE(vElemRef;"12<18")
```

このコードを実行すると、次のようになります:

```
 <Maths>
   <Postulate>12 < 18</Postulate>
</Maths>
```

```
  ` CDATA form
 DOM SET XML ELEMENT VALUE(vElemRef;"12<18";*)
```

このコードを実行すると、次のようになります:

```
<Maths>
   <Postulate><![CDATA[12 < 18]]></Postulate>
</Maths>
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます (例えば要素参照が無効な時など)。

XML SAX

- XML SAXコマンドの概要
- SAX ADD PROCESSING INSTRUCTION
- SAX ADD XML CDATA
- SAX ADD XML COMMENT
- SAX ADD XML DOCTYPE
- SAX ADD XML ELEMENT VALUE
- SAX CLOSE XML ELEMENT
- SAX GET XML CDATA
- SAX GET XML COMMENT
- SAX GET XML DOCUMENT VALUES
- SAX GET XML ELEMENT
- SAX GET XML ELEMENT VALUE
- SAX GET XML ENTITY
- SAX Get XML node
- SAX GET XML PROCESSING INSTRUCTION
- SAX OPEN XML ELEMENT
- SAX OPEN XML ELEMENT ARRAYS
- SAX SET XML DECLARATION

XML SAXコマンドの概要

このテーマには4DのXML SAXコマンドがまとめられています。

XMLに関する一般的な情報や、文字セット、DOMとSAXの相違点などについてはの節を参照してください。

SAXでXMLドキュメントを作成、開く、及び閉じる

SAXコマンドは4D標準のドキュメント参照を使用して動作します (DocRef, 時間型参照)。つまりSEND PACKET や Append documentのようなドキュメントを管理するコマンドとともにこれらのコマンドを使用できます。

プログラムによるXMLドキュメントの作成および開くことは、Create document や Open documentコマンドを使用して実行します。それ以降、これらのドキュメントに対するXMLコマンドの使用は、自動でエンコーディングなどのXMLメカニズムを適用します。例えば <?xml version="1.0" encoding="… encoding …" standalone = "no "?> は自動でドキュメントに書き込まれます。

Note: SAXコマンドで読み込まれるドキュメントはOpen document コマンドにおいて読み込みのみモードで開かれなければなりません。これにより、"標準"およびXMLドキュメントを同時に開くとき、4DとXercesライブラリの間でのコンフリクトが避けられます。読み書きモードで開いたドキュメントでSAX解析コマンドを実行すると、警告が表示され、解析を行うことは できません。

CLOSE DOCUMENT コマンドを使用してXMLドキュメントを閉じなければなりません。XML要素が開かれていると、それらは自動で閉じられます。

SAX ADD PROCESSING INSTRUCTION

SAX ADD PROCESSING INSTRUCTION (document ; statement)

引数	型		説明
document	DocRef	⇒	開かれたドキュメントの参照
statement	テキスト	⇒	ドキュメントに挿入するステートメント

説明

SAX ADD PROCESSING INSTRUCTIONコマンドは、documentで参照されるXMLドキュメントにXML処理命令statementを追加します。

処理命令を使用してアプリケーションのタイプや必要に応じて追加のパラメタを指定し、解析できない外部エンティティを処理できます。

コマンドはXMLに従ってデータステートメントをフォーマットします。しかしステートメント自身は解析されず、それが有効であることを確認するのは開発者の責任です。

例題

以下のコードにおいて:

```
 vtInstruct:="xml-stylesheet type="+Char(Quotes)+"text/xsl"+Char(Quotes)+
 "href="+Char(Quotes)+"style.xsl"+Char(Quotes)
 SAX ADD PROCESSING INSTRUCTION($DocRef;vtInstruct)
```

上記のコードはドキュメントに以下の行を書き込みます:

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

SAX ADD XML CDATA

SAX ADD XML CDATA (document ; data)

引数	型		説明
document	DocRef	→	開かれたドキュメントへの参照
data	BLOB, テキスト	→	ドキュメントのCDATAタグの間に挿入する テキストまたはBLOB

説明

SAX ADD XML CDATA コマンドは、documentで参照されるXMLドキュメントにテキストまたはBLOBのdataを追加します。このdataは自動で<![CDATA[と]]>の間におかれます。
CDATAセクションに含まれるテキストはXMLインタプリタにより無視されます。

dataの内容をエンコードしたい場合、BASE64 ENCODEコマンドを使用しなければなりません。この場合、もちろんdataにはBLOBを渡します。

このコマンドが正しく動作するためには、要素が開かれていなくてはなりません。そうでなければエラーが生成されます。

例題

XMLドキュメントに以下の行を挿入したいとします:

```
function matchwo(a,b)
{
if (a < b && a < 0) then
      {
      return 1
      }
else
      {
      return 0
      }
}
```

これを行うには、以下のコードを実行します:

```
 C_TEXT(vtMytext)
 ... ` place the text in the vtMytext variable here
 SAX ADD XML CDATA($DocRef;vtMytext)
```

結果は以下のようになります:

```
<![CDATA[
function matchwo(a,b)
{
if (a < b && a < 0) then
   {
   return 1
   }
else
   {
   return 0
   }
}
]]>
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

SAX ADD XML COMMENT

SAX ADD XML COMMENT (document ; comment)

引数	型		説明
document	DocRef	→	開かれたドキュメントの参照
comment	文字	→	追加するコメント

説明

SAX ADD XML COMMENT コマンドはdocumentで参照されるXMLドキュメントに、commentで指定したコメントを追加します。

XMLのコメントはXMLインタプリタが内容を解析しないテキストです。XMLコメントは <!-- と --> 文字の間に書かれなければなりません。

例題

以下のコードは:

```
vComment:="Created by 4D"
SAX ADD XML COMMENT($DocRef;vComment)
```

ドキュメントに以下のコードを書き込みます:

```
<!--Created by 4D-->
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

エラー管理

エラーが発生した場合、コマンドはエラー処理メソッドでとらえることのできるエラーを返します。

SAX ADD XML DOCTYPE

SAX ADD XML DOCTYPE (document ; docType)

引数	型		説明
document	DocRef	→	開いたドキュメントの参照
docType	文字	→	追加するDocType

説明

SAX ADD XML DOCTYPE コマンドは、documentで参照されるXMLドキュメントにdocType 引数で指定されたDocType文を追加します。

DocType文は書かれたXMLのタイプを示し、使用される Document Type Declaration (DTD) を指定するために使用されます。DocType文は通常以下の形式です: *<!DOCTYPE XML_type "DTD_address">*

例題

以下のコードは:

```
 vDocType:="SYSTEM Books \"Book.DTD\""
 SAX ADD XML DOCTYPE($DocRef;vDocType)
```

ドキュメントに以下の行を書き込みます:

```
<<!DOCTYPE SYSTEM Books "Book.DTD">
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、エラーが発生すると0が設定されます。

エラー管理

エラーが発生した場合、コマンドはエラー処理メソッドでとらえることのできるエラーを返します。

SAX ADD XML ELEMENT VALUE

SAX ADD XML ELEMENT VALUE (document ; data {; *})

引数	型		説明
document	DocRef	→	開いたドキュメントの参照
data	テキスト, 変数	→	ドキュメントに挿入するテキストまたは変数
*	演算子	→	指定時: 特別文字をエンコード 省略時: エンコードしない

説明

SAX ADD XML ELEMENT VALUEコマンドは、documentで参照されるXMLドキュメントにdataを変換せずに直接追加します。このコマンドは例えば電子メールのボディに添付ファイルを挿入するのと同等です。

dataには、直接文字列を、または4D変数を渡せます。変数の内容はXMLドキュメントに挿入される前にテキストに変換されます。
dataの内容をエンコードしたい場合BASE64 ENCODEコマンドを使用しなければなりません。この場合もちろんBLOBをdataに渡します。

XML SET OPTIONSコマンドのXML String encodingオプションにをXML Raw data渡すことでカレントプロセスにおいてこのメカニズムを無効にしない限り、デフォルトでコマンドはdata引数に含まれる特別文字 (< > " '...) をエンコードします。例:

```
XML SET OPTIONS($docRef;XML String encoding;XML Raw data)
```

この設定を行ったあと、**SAX ADD XML ELEMENT VALUE**を呼び出したとき文字のエンコーディングを強制したい場合、オプションの*引数を渡します。

このコマンドが正しく実行するには、要素が開かれていなければなりません。そうでなければエラーが生成されます。

例題

この例題は開かれたXML要素にwhitepaper.pdfファイルを挿入します:

```
C_BLOB(vBMyBLOB)
DOCUMENT TO BLOB("c:\\whitepaper.pdf";vBMyBLOB)
SAX ADD XML ELEMENT VALUE($DocRef;vBMyBLOB)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

SAX CLOSE XML ELEMENT

SAX CLOSE XML ELEMENT (document)

引数	型		説明
document	DocRef	→	開かれたドキュメントの参照

説明

SAX CLOSE XML ELEMENT コマンドは、documentで参照されるXMLドキュメントに、SAX OPEN XML ELEMENT コマンドを使用して開かれた最後の要素を閉じるのに必要な文を書き込みます。

このコマンドの利用は任意です。実際4DはXMLドキュメントが閉じられるときに、閉じられていない要素に必要な終了タグを自動で追加します。

例題

最後に開かれた要素が<Book>であるとき、以下のコードは:

```
SAX CLOSE XML ELEMENT($DocRef)
```

以下の行をドキュメントに書き込みます:

```
</Book>
```

SAX GET XML CDATA

SAX GET XML CDATA (document ; value)

引数	型		説明
document	DocRef	⇒	開いたドキュメントの参照
value	テキスト, BLOB	⇐	要素値

説明

SAX GET XML CDATAコマンドを使用して、document 引数で参照されるXMLドキュメント中に存在するXML要素のCDATA value を取得できます。このコマンドはXML CDATA SAX イベント内で呼び出さなければなりません。SAXイベントに関する詳細はSAX Get XML nodeコマンドの説明を参照してください。

32 KBを超えるデータを取得するにはテキスト型の変数をvalueに渡します (データベースはUnicodeモードで実行されていなければなりません)。

互換性に関する注記: 4D v12以降、base64でエンコードされたCDATAの内容は**SAX GET XML CDATA**コマンドにより自動でデコードされるようになりました。BASE64 DECODEコマンドを呼び出す必要はありません。

例題

以下のXMLコードがあります:

```
<RootElement>
   <Child>MyText<![CDATA[MyCData]]</Child>
</RootElement>
```

以下の4Dコードは “MyCData” を vTextData に返します:

```
C_BLOB(vData)
C_TEXT(vTextData)
SAX GET XML CDATA(DocRef;vData)
vTextData:=BLOB to text(vData;UTF8 C string)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

SAX GET XML COMMENT

SAX GET XML COMMENT (document ; comment)

引数	型		説明
document	DocRef	→	開いたドキュメントの参照
comment	文字	←	XMLコメント

説明

SAX GET XML COMMENT コマンドは、document 引数で参照されるXMLドキュメント中でXML Comment SAXイベントが生成された時、XMLコメントをcommentに返します。SAXイベントに関する詳細はSAX Get XML node コマンドの説明を参照してください。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

SAX GET XML DOCUMENT VALUES

SAX GET XML DOCUMENT VALUES (document ; encoding ; version ; standalone)

引数	型	説明
document	DocRef	→ 開いたドキュメントの参照
encoding	文字	← XMLドキュメント文字セット
version	文字	← XMLバージョン
standalone	ブール	← True = ドキュメントはスタンドアロン, そうない場合 False

説明

SAX GET XML DOCUMENT VALUES コマンドは、document 引数で参照されるXMLドキュメントのXMLヘッダから基本情報を取得します。

コマンドはエンコーディングのタイプ、バージョン、そして"スタンドアロン" プロパティをそれぞれencoding、version、そしてstandalone引数に返します。このコマンドはXML Start Documentイベント内で使用されなければなりません。SAXイベントに関する詳細はSAX Get XML node コマンドの説明を参照してください。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

SAX GET XML ELEMENT

SAX GET XML ELEMENT (document ; name ; prefix ; attrNames ; attrValues)

引数	型		説明
document	DocRef	→	開いたドキュメントの参照
name	文字	←	要素名
prefix	文字	←	名前空間
attrNames	文字配列	←	属性名
attrValues	文字配列	←	属性値

説明

SAX GET XML ELEMENT コマンドはdocument 引数で参照されるXMLドキュメントに現れる、name要素についてのさまざまな情報を返します。このコマンドはXML Start ElementまたはXML End Element SAXイベントで呼び出さなければなりません。XML End Elementの特定のケースでは、属性引数は処理されません。SAXイベントに関する詳細はSAX Get XML node コマンドの説明を参照してください。

name 引数には要素名を渡します。

prefix 引数には要素の名前空間が返されます。要素に名前空間がリンクされていない場合は空の文字列となります。

attrNames 配列にはターゲット要素の属性名リストが返されます。必要に応じてコマンドは配列を作成しサイズを調整します。

attrValues 配列にはターゲット要素の属性値が返されます。必要に応じてコマンドは配列を作成しサイズを調整します。

例題

以下のXMLコードがあります:

```
<RootElement>
   <Child Att1="111" Att2="222" Att3="333">MyText</Child>
</RootElement>
```

以下のコードが実行されると:

```
 SAX GET XML ELEMENT(DocRef;vName;vPrefix;tAttrNames;tAttrValues)
```

vNameには"Child"が、
vPrefixには""が、
tAttrNames{1} には "Att1"、
tAttrNames{2} には "Att2"、
tAttrNames{3} には "Att3"が
tAttrValues{1} には "111"、
tAttrValues{2} には "222"、
tAttrValues{3} には "333"が返されます。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

SAX GET XML ELEMENT VALUE

SAX GET XML ELEMENT VALUE (document ; value)

引数	型		説明
document	DocRef	→	開いた度kつ面との参照
value	テキスト, BLOB	←	要素値

説明

SAX GET XML ELEMENT VALUE コマンドは、document 引数で参照されるXMLドキュメント中に存在するXML要素の要素値をvalueに返します。このコマンドはXML DATA SAXイベントで呼び出さなければなりません。SAXイベントに関する詳細はSAX Get XML node コマンドの説明を参照してください。

TextまたはBLOB 型の変数をvalue 引数に渡します。BLOBを渡すと、コマンドは 自動でBase64のデコードを試みます。

例題

以下のXMLコードがあります:

```
<RootElement>
   <Child Att1="111" Att2="222" Att3="333">MyText</Child>
</RootElement>
```

以下のコードは"MyText"をvValueに返します:

```
 SAX GET XML ELEMENT VALUE(DocRef;vValue)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

SAX GET XML ENTITY

SAX GET XML ENTITY (document ; name ; value)

引数	型		説明
document	DocRef	→	開いたドキュメントの参照
name	文字	←	実体名
value	文字	←	実体値

説明

SAX GET XML ENTITYコマンドを使用して、document 引数で参照されるXMLドキュメント中に存在するXML実体の名前と値をnameとvalueに取得できます。このコマンドはXML Entity SAXイベントで呼び出されなければなりません。SAXイベントに関する詳細はSAX Get XML node コマンドの説明を参照してください。

例題

以下のXMLコードがあります:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE body [
   <!ELEMENT body (element*)>
   <!ELEMENT element (#PCDATA)>
   <!ENTITY name "Replacement">
]>
<body>
   <element>Entity updated by &name;</element>
</body>
```

以下のコードを実行するとvNameに"name"が、vValueに"Replacement"が返されます。

```
 SAX GET XML ENTITY(DocRef;vName;vValue)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されてエラーが生成されます。

SAX Get XML node

SAX Get XML node (document) -> 戻り値

引数	型		説明
document	DocRef	→	開いたドキュメントの参照
戻り値	倍長整数	↩	関数から返されたイベント

説明

SAX Get XML node コマンドは、document で参照されるXMLドキュメントが解析されている間、SAXイベントのタイプを示す倍長整数値を返します。

返されるイベントは""テーマの定数にあります:

定数	型	値
XML CDATA	倍長整数	7
XML Comment	倍長整数	2
XML DATA	倍長整数	6
XML End Document	倍長整数	9
XML End Element	倍長整数	5
XML Entity	倍長整数	8
XML Processing Instruction	倍長整数	3
XML Start Document	倍長整数	1
XML Start Element	倍長整数	4

例題

以下の例題でイベントの処理方法を示します:

```
DocRef:=Open document("";"xml";Read Mode)
If(OK=1)
   Repeat
      MyEvent:=SAX Get XML node(DocRef)
      Case of
         :(MyEvent=XML Start Document)
            DoSomething
         :(MyEvent=XML Comment)
            DoSomethingElse
      End case
   Until(MyEvent=XML End Document)
   CLOSE DOCUMENT(DocRef)
End if
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKには1が、そうでなければ0が設定されエラーが生成されます。

SAX GET XML PROCESSING INSTRUCTION

SAX GET XML PROCESSING INSTRUCTION (document ; name ; value)

引数	型		説明
document	DocRef	→	開いたドキュメントの参照
name	文字	←	命令名
value	文字	←	命令値

説明

SAX GET XML PROCESSING INSTRUCTION コマンドは、document 引数で参照されるXMLドキュメント中で処理されるXML命令の名前と値をnameとvalueに返します。このコマンドはXML Processing Instructionイベントで呼び出されなければなりません。SAXイベントに関する詳細はSAX Get XML node コマンドの説明を参照してください。

例題

以下のXMLコードがあります:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited with XML Spy v3.0.7 NT (http://www.xmlspy.com) by Myself (4D SA)-->
<?PI TextProcess?>
<!DOCTYPE RootElement SYSTEM "ParseTest.dtd">
```

以下のコードを実行するvNameに"PI"が、vValueに"TextProcess"が返されます:

```
 SAX GET XML PROCESSING INSTRUCTION($DocRef;vName;vValue)
```

SAX OPEN XML ELEMENT

SAX OPEN XML ELEMENT (document ; tag {; attribName ; attribValue} {; attribName2 ; attribValue2 ; ... ; attribNameN ; attribValueN})

引数	型		説明
document	DocRef	⇒	開いたドキュメントの参照
tag	文字	⇒	開く要素の名前
attribName	文字	⇒	属性名
attribValue	文字	⇒	属性値

説明

SAX OPEN XML ELEMENT コマンドを使用して、document で参照されるXMLドキュメント中に新しい要素およびオプションで属性とその値を追加できます。

追加された要素はドキュメント中で開かれた状態です (終了タグは追加されません)。作成した要素を閉じるには、以下のいずれかの方法を使用します:

- SAX CLOSE XML ELEMENT コマンドを使用する
- XMLドキュメントを閉じる。4Dは自動で必要なXML終了タグを追加します。

tagには、作成する要素の名前を渡します。この名前には文字、数字 “.”, “-”,“_” aや “:”.などの文字のみを渡せます。名前はXMLの仕様に従っていなければなりません。無効な文字がtagに渡されると、エラーが生成されます。

オプションでattribNameとattribValue引数を使用し、(変数、フィールド、またはリテラル値で) コマンドに1つ以上の属性名/値のペアを渡すことができます。

例題

以下のコードは:

```
vElement:="Book"
SAX OPEN XML ELEMENT($DocRef;vElement)
```

ドキュメントに以下の行を書き込みます:

```
<Book
```

エラー管理

tagに無効な文字が渡されるとエラーが生成されます。

SAX OPEN XML ELEMENT ARRAYS

SAX OPEN XML ELEMENT ARRAYS (document ; tag {; attribNamesArray ; attribValuesArray} {; attribNamesArray2 ; attribValuesArray2 ; ... ; attribNamesArrayN ; attribValuesArrayN})

引数	型		説明
document	DocRef	⇒	開いたドキュメントの参照
tag	文字	⇒	開く要素の名前
attribNamesArray	文字配列	⇒	属性名配列
attribValuesArray	文字配列, 倍長整数配列, 日付配列, 実数配列, ピクチャー配列, ブール配列	⇒	属性値配列

説明

SAX OPEN XML ELEMENT ARRAYS コマンドコマンドを使用して、document で参照されるXMLドキュメント中に新しい要素およびオプションで属性とその値を配列で指定して追加できます。

配列をサポートすること以外 (後述参照)、このコマンドはSAX OPEN XML ELEMENTと同じです。処理に関する情報はこのコマンドの説明を参照してください。

SAX OPEN XML ELEMENT ARRAYS は日付、数値、ブール、そしてピクチャ型の配列をattribValuesArray引数を受け入れます。4Dは自動で必要な変換を行います。XML SET OPTIONSコマンドを使用してこれらの変換を設定できます。

オプションでSAX OPEN XML ELEMENT ARRAYS コマンドは、attribNamesArrayとattribValuesArray引数に属性名と属性値のペアを配列形式で渡すことができます。

配列は事前に作成され、属性名と属性値のペアで構成されていなければなりません。複数の配列を渡すことができ、それぞれの配列に複数の配列要素を含めることができます。

例題

以下のメソッドは:

```
ARRAY TEXT(tAttrNames;2)
ARRAY TEXT(tAttrValues;2)
vElement:="Book"
tAttrNames{1}:="Font"
tAttrValues{1}:="Arial"
tAttrNames{2}:="Style"
tAttrValues{2}:="Bold"
SAX OPEN XML ELEMENT ARRAYS($DocRef;vElement;tAttrNames;tAttrValues)
```

以下の行をドキュメントに書き込みます:

```
<Book Font="Arial" Style="Bold">
```

SAX SET XML DECLARATION

SAX SET XML DECLARATION (document ; encoding {; standalone {; indentation}})

引数	型		説明
document	DocRef	⇒	開いたドキュメントの参照
encoding	文字	⇒	XMLドキュメント文字セット
standalone	ブール	⇒	True = ドキュメントはスタンドアロン False (デフォルト) = ドキュメントはスタンドアロンではない
indentation	ブール	⇒	True (デフォルト) = ドキュメントをインデントする False = ドキュメントをインデントしない

説明

SAX SET XML DECLARATION コマンドはdocument で参照されるXMLドキュメントを、引数に渡された値を使用して初期化します。これらのパラメタはエンコーディング、スタンドアロン、およびドキュメントをインデントするかを指定するために使用します。

- encoding: ドキュメントで使用される文字セットを指定するために使用します。コマンドが呼び出されない場合のデフォルトはUTF-8文字セットです。
- standalone: ドキュメントがスタンドアロンか (True)、あるいは他のファイルや外部リソースを必要とするか (False) を示します。コマンドが呼び出されないか引数が省略された場合のデフォルトはFalseです。

互換性に関する注意: indentation 引数は以前のバージョンの4Dとの互換性を保つために保持されていますが、4D v12よりその利用は推奨されません。今後、ドキュメントのインデントを設定するには、XML SET OPTIONSコマンドの利用が強く推奨されます。

このコマンドはドキュメントごとに一回、最初のXML設定コマンドの前に、呼び出さなければなりません。そうでなければエラーが生成されます。

例題

以下のコードを実行すると:

```
 SAX SET XML DECLARATION($DocRef;"UTF-16";True)
```

ドキュメントに以下の行が書き込まれます:

```
<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
```

インポート＆エクスポート

- EXPORT DATA
- EXPORT DIF
- EXPORT ODBC
- EXPORT SYLK
- EXPORT TEXT
- IMPORT DATA
- IMPORT DIF
- IMPORT ODBC
- IMPORT SYLK
- IMPORT TEXT

EXPORT DATA

EXPORT DATA (fileName {; project {; *}})

引数	型	説明
fileName	文字	→ エクスポートファイルのフルパス名
project	テキスト変数, BLOB変数	→ エクスポートプロジェクト ← エクスポートプロジェクトの新しい内容 (*引数指定時)
*	演算子	→ エクスポートダイアログを表示し プロジェクトの内容を更新

説明

EXPORT DATAコマンドは、データをfileNameファイルに書き出します。4Dからは 以下のフォーマットでデータの書き出しを行えます: テキスト、固定長テキスト、XML、SYLK、DIF、DBF(dBase)および4Dフォーマット

fileNameに空の文字列を渡すと、**EXPORT DATA**は標準のファイルを保存ダイアログボックスを表示して、書き出すファイルの名前、タイプおよび場所をユーザーが指定できるようにします。ダイアログボックスが受け入れられると、Documentシステム変数にファイルパスがセットされます。ユーザが**キャンセル**をクリックすると、コマンドの実行は停止されて、システム変数OKは0になります。

オプション引数projectを使用すると、データ書き出しにプロジェクトを使用できます。この引数を渡すと、書き出しはユーザーの操作を経ることなく直接行われます (後述の * 引数を指定しない限り)。この引数を渡さないと、書き出しダイアログボックスが表示されます。ユーザーは書き出しパラメターを指定するか、既存の書き出しプロジェクトをロードできます。

書き出しプロジェクトには、書き出しテーブルやフィールド、区切り符号のような書き出しに関するすべてのパラメターが含まれています。project引 数にはXMLで記述されたテキスト変数、定義済みのDOM要素への参照を格納したテキスト変数、またはBLOBのいずれかを渡せます。プロジェクトはプログラム (XMLフォーマットのプロジェクトのみ)、または書き出しダイアログボックスで事前に定義済みのパラメターをロードすることで作成できます。後者の場 合、2つの方法を利用できます:

- 空のプロジェクト引数およびオプションの * 付きで**EXPORT DATA**コマンドを使用し、ダイアログでの設定内容をproject引数に受け取って、それをテキストまたはBLOBフィールドに格納します (後述)。この方法ではプロジェクトをデータファイルに保存できます。
- プロジェクトをディスクに保存し、DOM Parse XML sourceコマンド等を使用してロードします。そしてその参照をproject引数に渡します。

互換性に関する注意: 4D v12より、読み込みプロジェクトはXMLにエンコードされるようになりました。4Dは以前のバージョンで作成された書き出しプロジェクト (BLOB形式) を開くことができます。しかし4D v12以降で作成されたプロジェクトをv11以前で開くことはできません。書き出しファイルを扱う際は、以降テキスト変数を使用することをお勧めします。

オプションの引数 * が指定されていれば、projectに定義されたパラメターと共にデータ書き出しダイアログボックスを表示します。これは、定義済みのプロジェクトを使用しつつ、いくつかのパラメターを変更できるようにするものです。さらに、データ書き出しダイアログボックスを閉じた後に、project引数には、新しいプロジェクトのパラメターが格納され、この新しいプロジェクトをBLOBフィールドやディスク等に保存することができます。

データ書き出しが正常に終了すると、システム変数OKは1になります。

例題 1

この例題ではバイナリーフォーマットでデータを書き出すために**EXPORT DATA**コマンドを使用する方法を示します。

- このメソッドはループ中ですべてのテーブルに対しExportBinaryメソッドを呼び出します:

```
C_TEXT($ExportPath)
C_LONGINT($i)
$ExportPath:=Select folder("書き出しフォルダーを選択:")
If(OK=1)
   For($i;1;Get last table number)
      If(Is table number valid($i))
         ExportBinary(Table($i);$ExportPath+Table name($i);True)
      End if
   End for
End if
```

- ExportBinary メソッドのコードは以下の通りです:

```
C_POINTER($1) //テーブルポインター
C_TEXT($2) //書き出し先ファイルのパス名
C_BOOLEAN($3) //True: すべてのレコードを書き出し
C_LONGINT($i)
C_TEXT($ref)
$ref:=DOM Create XML Ref("settings-import-export")
 // "$1"テーブルの全レコードまたはカレントセレクションを'4D'バイナリーフォーマットで書き出す。
DOM SET XML ATTRIBUTE($ref;"table_no";Table($1);"format";"4D";"all_records";$3)
 // 書き出すフィールドの指定
For($i;1;Get last field number($1))
   If(Is field number valid($1;$i))
      $elt:=DOM Create XML element($ref;"field";"table_no";Table($1);"field_no";$i)
   End if
End for
EXPORT DATA($2;$ref)
If(OK=0)
   ALERT(Table name($1)+"テーブルを書き出し中にエラーが発生しました。")
End if
DOM CLOSE XML($ref)
```

例題 2

この例題は、空のプロジェクトを作成し、ユーザが書き出しダイアログボックスで設定した各パラメターを保存します:

```
C_TEXT($exportParams)
EXPORT DATA("DocExport.txt";$exportParams;*) // 書き出しダイアログボックスを表示する
```

システム変数およびセット

標準のファイルを開くまたは書き出しダイアログボックスでユーザがキャンセルをクリックするとOKシステム変数は0に設定されます。書き出しが行われると1に設定されます。

EXPORT DIF

EXPORT DIF ({aTable ;} document)

引数	型		説明
aTable	テーブル	⇒	データを書き出すテーブル, または 省略した場合, デフォルトテーブル
document	文字	⇒	データが書き出されるDIFドキュメント

説明

EXPORT DIF コマンドは、カレントプロセスにおけるaTableのカレントセレクションのレコードをディスクに書き出します。このデータはdocumentに書き込まれます。documentは、WindowsまたはMacintoshの標準的なDIF形式のドキュメントです。

データの書き出し処理は、カレント出力フォームを介して実行します。このデータ書き出し処理は、出力フォーム上のフィールドや変数をその入力順序に 従って書き出します。このため、書き出したいフィールドと入力可能オブジェクトのみを含んだ出力フォームを使用する必要があります。書き出し用のフォーム にボタンやその他関係ないオブジェクトを配置しないでください。サブフォームオブジェクトは無視されます。

書き出されるレコードごとに、On Loadイベントがフォームメソッドに送られます。このイベントを利用して、書き出し用フォームで使用する変数を設定します。

document引数には、新規または既存のドキュメントファイルを指定することができます。documentが既存のドキュメントファイルと同じ名前の場合には、そのドキュメントファイルは上書きされます。また、documentにはボリューム名やフォルダ名等のパスを含めることもできます。空の文字列を渡すと、標準のファイルを保存ダイアログボックスが表示されます。ユーザがこのダイアログボックスをキャンセルすると、データ書き出し処理は中止され、システム変数OKには0がセットされます。

データの書き出し処理中には進捗インジケータが表示されます。インジケータの中止ボタンをクリックすると、処理を中断します。書き出しが正常に完了 すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には0がセットされます。インジケータを表示したくない場合には、MESSAGES OFFコマンドを使用してください。

デフォルトでは、コマンドはデフォルトでUTF-8文字セットを使用します。DIFフォーマットドキュメントは一般的にIBM437文字セットを使用するため、適切な文字セットを指定するためにUSE CHARACTER SETコマンドを使用する必要があるでしょう。

EXPORT DIF を使用する際、デフォルトのフィールド区切り文字はタブ文字 (コード 9) です。デフォルトのレコード区切り文字はキャリッジリターン (コード 13) です。これらの値は2つの システム変数、FldDelimit と RecDelimit に新しい値を代入することによって変更できます。ユーザはこれらのデフォルト値をデザインモードの書き出しダイアログで変更できます。テキストフィールドにはタブやキャリッジリターンを含めることができるので、これらの値を区切り文字として使用する際は注意が必要です。

例題

以下の例は、データをDIFドキュメントファイルに書き出します。まず、メソッドの最初で書き出しに使用する出力フォームを設定し、次にデータ書き出しを実行します:

```
FORM SET OUTPUT([People];"Export")
EXPORT DIF([People];"NewPeople.dif"")  ` "NewPeople.dif" ドキュメントに書き出しを行う
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

EXPORT ODBC

EXPORT ODBC (sourceTable {; project {; *}})

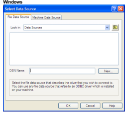
引数	型		説明
sourceTable	文字	⇒	ODBCデータソースにあるテーブルの名前
project	BLOB	⇒	書き出しプロジェクトの内容
		⇐	書き出しプロジェクトの新しい内容 (*が渡されていれば)
*	演算子	⇒	書き出しダイアログボックスとプロジェクト更新の表示

説明

EXPORT ODBCコマンドを使用して、外部ODBCソースのsourceTableテーブルへデータを書き出すことができます。

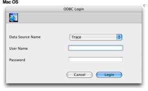
事前にSQL LOGINコマンドで接続を開かずにEXPORT ODBCコマンドを呼び出すと、ODBCデータソース選択ダイアログボックスが表示されます:

Windows



Select the file data source that describes the driver that you wish to connect to. You can use any file data source that refers to an ODBC driver which is installed on your machine.

Mac OS



Data Source Name: Trace
User Name
Password

ユーザがこのダイアログボックスで**キャンセル**ボタンをクリックすると、実行は中断されてシステム変数OKに0が代入されます。

Note: このコマンドは4D内部のSQLカーネルへの接続では使用できません。

オプションのproject引数を渡さない場合、4DはODBC書き出しダイアログを表示し、ユーザは行う処理を設定できます。このダイアログボックスに関する詳細はDesign Referenceを参照してください。

オプションのproject引数に有効なODBC書き出しプロジェクトを格納したBLOBを渡すと、ユーザによる操作を必要とせずに直接書き出しが実行されます。これを行うには、DOCUMENT TO BLOBコマンドを使用するなどして、ODBC書き出しダイアログからディスク上に保存されたプロジェクトを、あらかじめproject引数に渡すフィールドやBLOB変数にロードしておく必要があります。

また、空のproject引数とオプションの引数*を用いてEXPORT ODBCコマンドを呼び出した後で、引数projectをBLOBフィールド(後述)に保存することもできます。このソリューションにより、データファイルを用いてプロジェクトを保存し、ディスクからBLOBへのプロジェクトの保存フェーズを省略することができます。

Note: 空のプロジェクトの定義については、EXPORT DATAコマンドの例を参照してください。ただしODBC書き出しダイアログで作成されたプロジェクトは、4D標準の書き出しダイアログと互換がない点に留意してください。

オプションの引数*を指定した場合、projectに指定された設定(存在する場合)を用いて、ODBC書き出しダイアログボックスが表示されます。これにより、定義済みのプロジェクトを基に、いくつかのパラメータを変更することができます。さらに、その際はダイアログボックスを閉じた後に、新しいプロジェクトのパラメタがproject引数に格納されます。この後、それをBLOBフィールドやディスクファイルなどに保存できます。

システム変数およびセット

データソースの選択ダイアログや書き出し設定ダイアログでユーザがキャンセルボタンをクリックすると、OKシステム変数は0に設定されます。書き出しが正しく実行されると、OKシステム変数は1に設定されます。

EXPORT SYLK

EXPORT SYLK ({aTable ;} document)

引数	型		説明
aTable	テーブル	→	データを書き出すテーブル, または 省略した場合, デフォルトテーブル
document	文字	→	データが書き出されるSYLKドキュメント

説明

EXPORT SYLK コマンドは、カレントプロセスにおけるaTableのカレントセレクションのレコードをディスクに書き出します。このデータはdocumentに書き込まれます。documentは、WindowsまたはMacintoshの標準的なSYLK形式のドキュメントです。

データの書き出し処理は、カレント出力フォームを介して実行します。このデータ書き出し処理は、出力フォーム上のフィールドや変数をその入力順序に 従って書き出します。このため、書き出したいフィールドと入力可能オブジェクトのみを含んだ出力フォームを使用する必要があります。書き出し用のフォーム にボタンやその他関係ないオブジェクトを配置しないでください。サブフォームオブジェクトは無視されます。

書き出されるレコードごとに、On Loadイベントがフォームメソッドに送られます。このイベントを利用して、書き出し用フォームで使用する変数を設定します。

document引数には、新規または既存のドキュメントファイルを指定することができます。documentが既存のドキュメントファイルと同じ名前の場合には、そのドキュメントファイルは上書きされます。また、documentにはボリューム名やフォルダ名等のパスを含めることもできます。空の文字列を渡すと、標準のファイルを保存ダイアログボックスが表示されます。ユーザがこのダイアログボックスをキャンセルすると、データ書き出し処理は中止され、システム変数OKには0がセットされます。

データの書き出し処理中には進捗インジケータが表示されます。インジケータの中止ボタンをクリックすると、処理を中断します。書き出しが正常に完了 すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には0がセットされます。インジケータを表示したくない場合には、MESSAGES OFFコマンドを使用してください。

コマンドはデフォルトでUTF-8文字セットを使用します。SYLKフォーマットドキュメントは一般的にISO-8859-1文字セットを使用するため、適切な文字セットを指定するためにUSE CHARACTER SETコマンドを使用する必要があるでしょう。

EXPORT SYLKを使用する際、デフォルトのフィールド区切り文字はタブ文字です。デフォルトのレコード区切り文字はキャリッジリターン (コード 13) です。これらの値は2つのシステム変数、FldDelimit と RecDelimit に新しい値を代入して変更できます。ユーザはこれらのデフォルト値をデザインモードの書き出しダイアログで変更できます。テキストフィールドにはタブやキャリッジリターンを含めることができるので、これらの値を区切り文字として使用する際は注意が必要です。

例題

以下の例は、データをSYLKドキュメントファイルに書き出します。まず、メソッドの最初で書き出しに使用する出力フォームを設定し、次にデータ書き出しを実行します:

```
FORM SET OUTPUT([People];"Export")
EXPORT SYLK([People];"NewPeople.slk") ` "NewPeople.slk" ドキュメントに書き出しを行う
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

EXPORT TEXT

EXPORT TEXT ({aTable ;} document)

引数	型	説明
aTable	テーブル	⇒データを書き出すテーブル, または 省略した場合, デフォルトテーブル
document	文字	⇒データが書き出されるテキストドキュメント

説明

EXPORT TEXT コマンドは、カレントプロセスにおけるaTableのカレントセレクションのレコードをディスクに書き出します。このデータはdocumentに書き込まれます。documentは、WindowsまたはMacintoshのテキストドキュメントです。

データの書き出し処理は、カレント出力フォームを介して実行します。このデータ書き出し処理は、出力フォーム上のフィールドや変数をその入力順序に 従って書き出します。このため、書き出したいフィールドと入力可能オブジェクトのみを含んだ出力フォームを使用する必要があります。書き出し用のフォーム にボタンやその他関係ないオブジェクトを配置しないでください。サブフォームオブジェクトは無視されます。

書き出されるレコードごとに、On Loadイベントがフォームメソッドに送られます。このイベントを利用して、書き出し用フォームで使用する変数を設定します。

document引数には、新規または既存のドキュメントファイルを指定することができます。documentが既存のドキュメントファイルと同じ名前の場合には、そのドキュメントファイルは上書きされます。また、documentにはボリューム名やフォルダ名等のパスを含めることもできます。空の文字列を渡すと、標準のファイルを保存ダイアログボックスが表示されます。ユーザがこのダイアログボックスをキャンセルすると、データ書き出し処理は中止され、システム変数OKには0がセットされます。

データの書き出し処理中には進捗インジケータが表示されます。インジケータの中止ボタンをクリックすると、処理を中断します。書き出しが正常に完了 すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には0がセットされます。インジケータを表示したくない場合には、MESSAGES OFF コマンドを使用してください。

デフォルトで、コマンドはUTF-8文字コードを使用します。この文字コード設定は USE CHARACTER SET コマンドを使用することによって変更することができます。

EXPORT TEXT を使用する際、デフォルトのフィールド区切り文字はタブ文字 (コード 9) です。デフォルトのレコード区切り文字はキャリッジリターン (コード 13) です。これらのデフォルト設定は2つの システム変数、FldDelimit と RecDelimit に新しい値を代入することによって変更できます。ユーザはこれらのデフォルト値をデザインモードの書き出しダイアログで変更できます。テキストフィールドにはタブやキャリッジリターンを含めることができるので、これらの値を区切り文字として使用する際は注意が必要です。

例題

以下の例は、データをテキストドキュメントファイルに書き出します。まず、メソッドの最初で書き出しに使用する出力フォームを設定し、次に4Dの区切り文字変数を変更して、データ書き出しを実行します:

```
FORM SET OUTPUT([People];"Export")
FldDelimit:=27 ` フィールド区切り文字をEscape 文字にする
RecDelimit:=10 ` レコード区切り文字をLine Feed 文字にする
EXPORT TEXT([People];"NewPeople.txt") ` "NewPeople.txt" ドキュメントに書き出しを行う
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

IMPORT DATA

IMPORT DATA (fileName {; project {; *}})

引数	型		説明
fileName	文字	→	インポートファイルのフルパス名
project	テキスト変数, BLOB変数	→	インポートプロジェクト
		←	インポートプロジェクトの新しい内容 (*引数指定時)
*	演算子	→	インポートダイアログを表示し プロジェクトの内容を更新

説明

IMPORT DATAコマンドはデータをfileNameファイルから読み込みます。4Dは以下のフォーマットのデータ読み込みを行えます: テキスト、固定長テキスト、XML、SYLK、DIF、DBF (dBase) および4Dフォーマット

fileNameに空の文字列を渡すと、**IMPORT DATA**は標準のファイルを選択ダイアログボックスを表示して、ユーザが読み込みファイルの名前、タイプおよび場所を指定することができるようにします。ダイアログボックスが受け入れられると、Documentシステム変数にファイルパスが設定されます。ユーザが**キャンセル**をクリックすると、コマンドの実行は停止されて、システム変数OKは0になります。

オプション引数projectを使用すると、データ読み込みにプロジェクトを使用できます。この引数を渡すと、読み込みはユーザーの操作を経ることなく直接行われます (後述の * 引数を指定しない限り)。この引数を渡さないと、読み込みダイアログボックスが表示されます。ユーザーは読み込みパラメーターを指定するか、既存の読み込みプロジェクトをロードできます。

読み込みプロジェクトには、読み込むテーブルやフィールド、区切り符号のような読み込みに関するすべてのパラメーターが含まれています。project引数にはXMLで記述されたテキスト変数、定義済みのDOM要素への参照を格納したテキスト変数、またはBLOBのいずれかを渡せます。プロジェクトはプログラム (XMLフォーマットのプロジェクトのみ)、または読み込みダイアログボックスで事前に定義済みのパラメーターをロードすることで作成できます。後者の場合、2つの方法を利用できます:

- 空のプロジェクト引数およびオプションの * 付きで**IMPORT DATA**コマンドを使用し、ダイアログでの設定内容をproject引数に受け取って、それをテキストまたはBLOBフィールドに格納します (後述)。この方法ではプロジェクトをデータファイルに保存できます。
- プロジェクトをディスクに保存し、DOM Parse XML sourceコマンド等を使用してロードします。そしてその参照をproject引数に渡します。

互換性に関する注意: 4D v12より、読み込みプロジェクトはXMLにエンコードされるようになりました。4Dは以前のバージョンで作成された読み込みプロジェクト (BLOB形式) を開くことができます。しかし4D v12以降で作成されたプロジェクトをv11以前で開くことはできません。読み込みファイルを扱う際は、以降テキスト変数を使用することをお勧めします。

オプションの引数 * が指定されていれば、projectに定義されたパラメーターと共にデータ読み込みダイアログボックスを表示します。これは、定義済みのプロジェクトを使用しつつ、いくつかのパラメーターを変更できるようにするものです。さらにデータ読み込みダイアログボックスを閉じた後に、project引数には、新しいプロジェクトのパラメーターが格納され、この新しいプロジェクトをBLOBフィールドやディスク等に保存することができます。

データ読み込みが正常に終了すると、OKシステム変数は1になります。

Note: 空のプロジェクトを使用してプロジェクトを保存する例題はEXPORT DATA コマンドを参照してください。

システム変数およびセット

標準のファイルを保存または読み込みダイアログボックスでユーザがキャンセルをクリックするとOKシステム変数は0に設定されます。読み込みが行われると1に設定されます。

IMPORT DIF

IMPORT DIF ({aTable ;} document)

引数	型		説明
aTable	テーブル	⇒	データを読み込むテーブル, または 省略した場合, デフォルトテーブル
document	文字	⇒	データを読み込むDIFドキュメント

説明

IMPORT DIF コマンドは、WindowsまたはMacintoshの標準的なDIF形式のドキュメントdocumentから aTableにデータを読み込み、新しいレコードを作成します。

データの読み込み処理は、カレント入力フォームを介して実行します。このデータ読み込み処理は、入力フォーム上のフィールドや変数をそのレイヤに従って行われます。このため、フォーム中のテキストオブジェクト (フィールドや変数) の重なり順を注意深く設定する必要があります。最初にデータが読み込まれるオブジェクトは、フォームの最背面に置きます。読み込むフィールド数とフォーム 上のフィールドや変数の数が一致しない場合、余分なものは無視されます。サブフォームオブジェクトは無視されます。

Note: データが正しいオブジェクトに読み込まれることを確実にする方法のひとつは、読み込む最初のフィールドを選択しそれを最前面にし、引き続き順番にフィールドや変数を最前面に設定していき、インポートされるそれぞれのフィールドに対し、一つのフィールドまたは変数があることを確認しながら進めることです。

読み込まれるレコードごとに、On Validateイベントがフォームメソッドに送られます。このイベントを利用して、変数からフィールドにデータをコピーできます。

documentにはボリューム名やフォルダ名等のパスを含めることができます。空の文字列を渡すと、標準のファイルを開くダイアログボックスが表示されます。ユーザがこのダイアログボックスをキャンセルすると、データ読み込み処理は中止され、システム変数OKには0がセットされます。

データの読み込み処理中には進捗インジケータが表示されます。インジケータの中止ボタンをクリックすると、処理を中断します。しかし既に読み込まれたレコードは取り除かれません。読み込みが正常に完了すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には0がセットされます。インジケータを表示したくない場合には、 MESSAGES OFF コマンドを使用してください。

コマンドはデフォルトでUTF-8文字セットを使用します。DIFフォーマットドキュメントは一般的にIBM437文字セットを使用するため、適切な文字セットを指定するために USE CHARACTER SET コマンドを使用する必要があるでしょう。

IMPORT DIF を使用する際、デフォルトのフィールド区切り文字はタブ文字 (コード 9) です。デフォルトのレコード区切り文字はキャリッジリターン (コード 13) です。これらの値は2つのシステム変数、FldDelimit と RecDelimit に新しい値を代入して変更できます。ユーザはこれらのデフォルト値をデザインモードの読み込みダイアログで変更できます。テキストフィールドにはタブやキャリッジリターンを含めることができるので、これらの値を区切り文字として使用する際は注意が必要です。

例題

以下の例は、データをDIFドキュメントファイルから読み込みます。まず、メソッドの最初で読み込みに使用する入力フォームを設定し、次にデータ読み込みを実行します:

```
FORM SET INPUT([People];"Import")
IMPORT DIF([People];"NewPeople.dif") ` "NewPeople.dif" から読み込みを実行
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

IMPORT ODBC

IMPORT ODBC (sourceTable {; project {; *}})

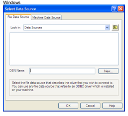
引数	型		説明
sourceTable	文字	→	ODBCデータソースのテーブル名
project	BLOB	→	読み込みプロジェクトの内容
		←	新しい読み込みプロジェクトの内容 (*が渡されていれば)
*	演算子	→	読み込みダイアログボックスの表示とプロジェクトの更新

説明

IMPORT ODBCコマンドを使用して、外部ODBCソースのsourceTableテーブルからデータを読み込むことができます。

事前にSQL LOGINコマンドで接続を開かずにIMPORT ODBCコマンドを呼び出すと、ODBCデータソース選択ダイアログボックスが表示されます:

Windows

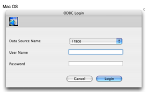


Look in: Data Sources

DSN Name:

Select the file data source that describes the driver that you wish to connect to. You can use any file data source that refers to an ODBC driver which is installed on your machine.

Mac OS



Data Source Name: Trace

User Name:

Password:

ユーザがこのダイアログボックスで**キャンセル**ボタンをクリックすると、実行は中断されてシステム変数OKに0が代入されます。

Note: このコマンドは4D内部のSQLカーネルへの接続では使用できません。

オプションのproject引数を渡さない場合、4DはODBC読み込みダイアログを表示し、ユーザは行う処理を設定できます。このダイアログボックスに関する詳細はDesign Referenceを参照してください。

オプションのproject引数に有効なODBC読み込みプロジェクトを格納したBLOBを渡すと、ユーザによる操作を必要とせずに直接読み込みが実行されます。これを行うにはODBC読み込みダイアログからディスク上に保存されたプロジェクトを、DOCUMENT TO BLOBコマンドを使用するなどしてあらかじめproject引数に渡すフィールドやBLOB変数にロードしておく必要があります。

また、空のproject引数とオプションの引数*を用いてIMPORT ODBCコマンドを呼び出した後で、引数projectをBLOBフィールド(後述)に保存することもできます。このソリューションにより、データファイルを用いてプロジェクトを保存し、ディスクからBLOBへのプロジェクトの保存フェーズを省略することができます。

Note: 空のプロジェクトの定義については、EXPORT DATAコマンドの例を参照してください。ただしODBC読み込みダイアログで作成されたプロジェクトは、4D標準の読み込みダイアログと互換がない点に留意してください。

オプションの引数*を指定した場合、projectに指定された設定(存在する場合)を用いて、ODBC読み込みダイアログボックスが表示されます。これにより、定義済みのプロジェクトを基に、いくつかのパラメータを変更することができます。さらに、その際はダイアログボックスを閉じた後に、新しいプロジェクトのパラメタがproject引数に格納されます。この後、それをBLOBフィールドやディスクファイルなどに保存できます。

システム変数およびセット

データソース選択ダイアログや読み込み設定ダイアログでユーザがキャンセルをクリックすると、システム変数OKは0に設定されます。読み込みが正しく実行されるとシステム変数OKは0に設定されます。

IMPORT SYLK

IMPORT SYLK ({aTable ;} document)

引数	型		説明
aTable	テーブル	⇒	データを読み込むテーブル, または 省略した場合, デフォルトテーブル
document	文字	⇒	データを読み込むSYLKドキュメント

説明

IMPORT SYLK コマンドは、WindowsまたはMacintoshの標準的なSYLK形式のドキュメントdocumentから aTableにデータを読み込み、新しいレコードを作成します。

データの読み込み処理は、カレント入力フォームを介して実行します。このデータ読み込み処理は、入力フォーム上のフィールドや変数をそのレイヤに従って行われます。このため、フォーム中のテキストオブジェクト (フィールドや変数) の重なり順を注意深く設定する必要があります。最初にデータが読み込まれるオブジェクトは、フォームの最背面に置きます。読み込むフィールド数とフォーム 上のフィールドや変数の数が一致しない場合、余分なものは無視されます。サブフォームオブジェクトは無視されます。

Note: データが正しいオブジェクトに読み込まれることを確実にする方法のひとつは、読み込む最初のフィールドを選択しそれを最前面にし、引き続き順番にフィールドや変数を最前面に設定していき、インポートされるそれぞれのフィールドに対し、一つのフィールドまたは変数があることを確認しながら進めることです。

読み込まれるレコードごとに、On Validateイベントがフォームメソッドに送られます。このイベントを利用して、変数からフィールドにデータをコピーできます。

documentにはボリューム名やフォルダ名等のパスを含めることができます。空の文字列を渡すと、標準のファイルを開くダイアログボックスが表示されます。ユーザがこのダイアログボックスをキャンセルすると、データ読み込み処理は中止され、システム変数OKには0がセットされます。

データの読み込み処理中には進捗インジケータが表示されます。インジケータの中止ボタンをクリックすると、処理を中断します。しかし既に読み込まれたレコードは取り除かれません。読み込みが正常に完了すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には0がセットされます。インジケータを表示したくない場合には、 MESSAGES OFF コマンドを使用してください。

コマンドはデフォルトでUTF-8文字セットを使用します。SYLKフォーマットドキュメントは一般的にISO-8859-1文字セットを使用するため、適切な文字セットを指定するために USE CHARACTER SET コマンドを使用する必要があるでしょう。

IMPORT SYLK を使用する際、デフォルトのフィールド区切り文字はタブ文字 (コード 9) です。デフォルトのレコード区切り文字はキャリッジリターン (コード 13) です。これらの値は2つのシステム変数、FldDelimit と RecDelimit に新しい値を代入して変更できます。ユーザはこれらのデフォルト値をデザインモードの読み込みダイアログで変更できます。テキストフィールドにはタブやキャリッジリターンを含めることができるので、これらの値を区切り文字として使用する際は注意が必要です。

例題

以下の例は、データをSYLKドキュメントファイルから読み込みます。まず、メソッドの最初で読み込みに使用する入力フォームを設定し、次にデータ読み込みを実行します:

```
FORM SET INPUT([People];"Import")
IMPORT SYLK([People];"NewPeople.slk") ` "NewPeople.slk" から読み込みを実行
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

IMPORT TEXT

IMPORT TEXT ({aTable ;} document)

引数	型		説明
aTable	テーブル	⇒	データを読み込むテーブル, または 省略した場合, デフォルトテーブル
document	文字	⇒	データを読み込むテキストドキュメント

説明

IMPORT TEXT コマンドは、WindowsまたはMacintoshの標準的なテキスト形式のドキュメントdocumentから aTable にデータを読み込み、新しいレコードを作成します。

データの読み込み処理は、カレント入力フォームを介して実行します。このデータ読み込み処理は、入力フォーム上のフィールドや変数をそのレイヤに従って行われます。このため、フォーム中のテキストオブジェクト (フィールドや変数) の重なり順を注意深く設定する必要があります。最初にデータが読み込まれるオブジェクトは、フォームの最背面に置きます。読み込むフィールド数とフォーム 上のフィールドや変数の数が一致しない場合、余分なものは無視されます。サブフォームオブジェクトは無視されます。

Note: データが正しいオブジェクトに読み込まれることを確実にする方法のひとつは、読み込む最初のフィールドを選択しそれを最前面にします。引き続き順番にフィールドや変数を最前面に設定していきます。

読み込まれるレコードごとに、On Validateイベントがフォームメソッドに送られます。このイベントを利用して、変数からフィールドにデータをコピーできます。

document にはボリューム名やフォルダ名等のパスを含めることができます。空の文字列を渡すと、標準のファイルを開くダイアログボックスが表示されます。ユーザがこのダイアログボックスをキャンセルすると、データ読み込み処理は中止され、システム変数OKには0がセットされます。

データの読み込み処理中には進捗インジケータが表示されます。インジケータの中止ボタンをクリックすると、処理を中断します。しかし既に読み込まれたレコードは取り除かれません。読み込みが正常に完了すると、システム変数OKに1がセットされ、エラーが発生、または処理が中断された場合には0がセットされます。インジケータを表示したくない場合には、MESSAGES OFF コマンドを使用してください。

Unicodeモード (標準モード) では、コマンドはデフォルトでUTF-8文字セットを使用します。USE CHARACTER SET コマンドを使用してこの文字セットを変更できます。

IMPORT TEXT を使用する際、デフォルトのフィールド区切り文字はタブ文字 (コード 9) です。デフォルトのレコード区切り文字はキャリッジリターン (コード 13) です。これらの値は2つのシステム変数 FldDelimit と RecDelimit に新しい値を代入して変更できます。ユーザはこれらのデフォルト値をデザインモードの読み込みダイアログで変更できます。テキストフィールドにはタブやキャリッジリターンを含めることができるので、これらの値を区切り文字として使用する際は注意が必要です。

例題

以下の例は、データをテキストドキュメントファイルから読み込みます。まず、メソッドの最初で読み込みに使用する入力フォームを設定し、次に4Dの区切り文字変数を変更して、データ読み込みを実行します:

```
FORM SET INPUT([People];"Import")
FldDelimit:=27 ` フィールド区切り文字をEscape 文字にする
RecDelimit:=10 ` レコード区切り文字をLine Feed 文字にする
IMPORT TEXT([People];"NewPeople.txt") ` "NewPeople.txt" から読み込みを実行
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

ウィンドウ

- ウィンドウの管理
- ウィンドウタイプ
- CLOSE WINDOW
- Current form window
- DRAG WINDOW
- ERASE WINDOW
- Find window
- Frontmost window
- GET WINDOW RECT
- Get window title
- HIDE WINDOW
- MAXIMIZE WINDOW
- MINIMIZE WINDOW
- Next window
- Open external window
- Open form window
- Open window
- REDRAW WINDOW
- RESIZE FORM WINDOW
- SET WINDOW RECT
- SET WINDOW TITLE
- SHOW WINDOW
- Window kind
- WINDOW LIST
- Window process

ウィンドウの管理

ウィンドウはユーザに情報を表示するために使用されます。主たる用途は3つ: データの入力、データの表示、そしてメッセージとダイアログによるユーザへの通知です。

ウィンドウは常に少なくとも1つ開かれています。必要に応じてスクロールバーが追加され、ユーザはウィンドウより大きなフォームをスクロールでき ます。デザインモードでは、このウィンドウはレコードリスト (出力フォーム)、またはデータ入力画面 (入力フォーム) を表示します。アプリケーションモードでは、このウィンドウはスプラッシュスクリーン (カスタマイズされた画像)を表示します。

アプリケーションプロセスでメニューコマンドを実行すると、フォームを表示するコマンドによりスプラッシュスクリーンがデータに置き換えられることがあります。コマンドが実行を終了すると、デフォルトでスプラッシュスクリーンが再び表示されます。

WinRef

さまざまなタイプのカスタムウィンドウを Open window や Open form window コマンドで開くことができます (ウィンドウタイプ の節参照)。これらのコマンドで開かれるすべてのウィンドウは**WinRef** 式で参照されます。WinRef は開かれたウィンドウの倍長整数タイプのユニークIDです。カスタムウィンドウに対して動作するコマンドはWinRef 引数を受け取ります。

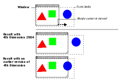
カスタムウィンドウが必要なくなったら、CLOSE WINDOW コマンドを使用、またはコントロールメニューボックス (Windows) やクローズボックス(Mac)があればそれをクリックして、ウィンドウを閉じます。

いくつかのコマンドは独自のウィンドウを開きます。QR REPORT 、PRINT LABEL などはウィンドウを開き、それが最前面のウィンドウになります。

新しく開始されたプロセスの中で実行されるプロセスメソッドでウィンドウを開いていない場合、4Dはフォームが表示されるときにデフォルトのウィンドウを開きます。

サイドプッシャー

ウィンドウの右端および下端はデフォルトでプッシャースプリッタになっています。つまりウィンドウ端の右や下にあるオブジェクトは、ウィンドウのサイズ変更に伴い自動で右方向や下方向に移動させられます:


Window
Form limits
Resize corner is moved
Result with 4th Dimension 2004
Result with an earlier version of 4th Dimension

このメカニズムを使用して、エクスプローラのような伸縮ウィンドウを管理できます (FORM SET SIZE コマンドの例題参照)。

Note: これはスクロールバーを持つウィンドウでは動作しません。

ウィンドウの座標と"right-to-left"モード

ウィンドウ管理コマンドで、ウィンドウの座標はウィンドウ/スクリーンの左上を起点に表されます。
しかし"right-to-left"モードが有効になっているアプリケーションでは、座標は反転し、起点はウィンドウ/スクリーンの右上となります。したがって、このモードでは以下のコマンドで使用される水平座標も反転させなければなりません。

Open window
Open form window
Open external window
GET WINDOW RECT
SET WINDOW RECT
Find window

Note: "right-to-left"モードに関する詳細は、Design Referenceマニュアルと SET DATABASE PARAMETER コマンドの説明を参照してください。

ウィンドウタイプ

Open windowコマンドで開くウィンドウのタイプを指定するために、以下の定義済み定数を使用できます:

定数	型	値	コメント
Plain no zoom box window	倍長整数	0	
Modal dialog box	倍長整数	1	
Plain dialog box	倍長整数	2	フローティング可
Alternate dialog box	倍長整数	3	フローティング可
Plain fixed size window	倍長整数	4	
Movable dialog box	倍長整数	5	フローティング可
Plain window	倍長整数	8	
Round corner window	倍長整数	16	
Pop up window	倍長整数	32	
Sheet window	倍長整数	33	
Resizable sheet window	倍長整数	34	
Palette window	倍長整数	1984	フローティング可

フローティングウィンドウ

これらの定数の一つをOpen windowに渡すと、通常のウィンドウが開かれます。フローティングウィンドウを開くには負数のウィンドウタイプをOpen windowに渡します。
フローティングウィンドウの主な特徴は、ユーザが他のウィンドウをクリックしても常に最前面に居続けるということです。フローティングウィンドウは一般に恒久的な情報やツールバーを表示するために使用されます。

モーダルウィンドウ

モーダルウィンドウは、ユーザのアクションをそのウィンドウに限定したい場合に使用します。モーダルウィンドウが表示されている間、メニューコマンドや他のアプリケーションウィンドウはアクセスできなくなります。モーダルウィンドウを閉じるには、ユーザはウィンドウを受け入れるか、キャンセルするか、提供される選択肢を選択しなければなりません。警告ダイアログは典型的なモーダルウィンドウの例です。
4Dにおいて、タイプ1と5がモーダルウィンドウです。

Note: モーダルウィンドウは常に最前面に居続けます。結果としてモーダルウィンドウが非モーダルウィンドウを呼び出すと、後者のウィンドウはモーダルウィンドウの後ろに開かれます。このような処理は避けなければなりません。
他方、モーダルウィンドウが他のモーダルウィンドウを呼び出す際は、後者のウィンドウが最前面に開かれます。

説明

以下に、Windows (左) およびMacintosh (右) のウィンドウタイプを示します。

Plain fixed size window (4)

- タイトルを持てる: はい
- クローズボックスまたは同等の機能を持てる: はい
- リサイズ可能: Macintosh上でいいえ
- 最小化/最大化、またはズーム可: いいえ
- スクロールバー: はい and いいえ
- 利用シーン: ADD RECORD(...;...*) または同等のデータ入力

Modal dialog box (1)

- タイトルを持てる: いいえ
- クローズボックスまたは同等の機能を持てる: いいえ
- リサイズ可能: いいえ
- 最小化/最大化、またはズーム可: いいえ
- スクロールバー: いいえ
- 利用シーン: DIALOG, ADD RECORD(...;...;*) または同等機能
- このタイプのウィンドウはモーダルです

Plain no zoom box window (0)

- タイトルを持てる: はい
- クローズボックスまたは同等の機能を持てる: はい
- リサイズ可能: はい
- 最小化/最大化、またはズーム可: Macintosh上でいいえ
- スクロールバー: はい
- 利用シーン: スクロールバーを持つデータ入力, DISPLAY SELECTION, MODIFY SELECTION, 他.

Plain window (8)

- タイトルを持てる: はい
- クローズボックスまたは同等の機能を持てる: はい
- リサイズ可能: はい
- 最小化/最大化、またはズーム可: はい
- スクロールバー: はい
- 利用シーン: スクロールバーを持つデータ入力, DISPLAY SELECTION, MODIFY SELECTION, 他.

Movable dialog box (5)

- タイトルを持てる: はい
- クローズボックスまたは同等の機能を持てる: いいえ
- リサイズ可能: いいえ
- 最小化/最大化、またはズーム可: いいえ
- スクロールバー: いいえ
- 利用シーン: DIALOG, ADD RECORD(...;...;*) または同等機能
- このタイプのウィンドウはモーダルですが、移動でき、フローティングのように使用できます

Alternate dialog box (3)

- タイトルを持てる: いいえ
- クローズボックスまたは同等の機能を持てる: いいえ
- リサイズ可能: いいえ
- 最小化/最大化、またはズーム可: いいえ
- スクロールバー: いいえ
- 利用シーン: DIALOG, ADD RECORD(...;...;*) または同等機能
- フローティングウィンドウとして使用されない場合、このタイプはモーダルです

Plain dialog box (2)

- タイトルを持てる: いいえ
- クローズボックスまたは同等の機能を持てる: いいえ
- リサイズ可能: いいえ
- 最小化/最大化、またはズーム可: いいえ
- スクロールバー: いいえ
- 利用シーン: DIALOG, ADD RECORD(...;...;*) または同等機能, スプラッシュスクリーン
- フローティングウィンドウとして使用されない場合、このタイプはモーダルです

Palette window (1984 {+ 1} {+ 2} {+ 4} {+ 8})

Open windowを呼び出すとき、以下の定数をPalette window定数に加算して、ウィンドウの振る舞いを変更できます:

定数	型	値
Has grow box	倍長整数	4
Has highlight	倍長整数	1
Has window title	倍長整数	2
Has zoom box	倍長整数	8

- タイトルを持てる: はい (Has window title が指定されていれば)
- クローズボックスまたは同等の機能を持てる: はい
- リサイズ可能: はい (Has grow box が指定されていれば)
- 最小化/最大化、またはズーム可: はい (Has zoom box が指定されていれば)
- スクロールバー: はい (Has grow box が指定されていれば)
- 利用シーン: DIALOGやDISPLAY SELECTIONによるフローティングウィンドウ (データ入力なし)

Round corner window (16)

- タイトルを持てる: はい
- クローズボックスまたは同等の機能を持てる: はい
- リサイズ可能: Macintosh上でいいえ
- 最小化/最大化、またはズーム可: いいえ
- スクロールバー: Macintosh上でいいえ
- 利用シーン: まれ (廃止)

Sheet window (33) と Resizable sheet window (34)

シートウィンドウはMac OS X特有のものです。これらのウィンドウはタイトルバーからドロップダウンし、メインウィンドウの上、自動的に中央に表示されます。プロパティはモーダルダイアログと互換があります。一般的にこのタイプのウィンドウは、主たるウィンドウ中で発生するアクションに直接関連するアクションを行うために使用されます。

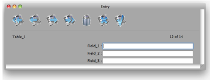
- Mac OS X上で、最後に開かれたウィンドウが表示されていてまたフォームを表示しているときにのみ、シートウィンドウを作成できます。
- このコマンドは以下の場合、タイプ33の代わりにタイプ 1 (Modal dialog box) のウィンドウを、タイプ34の代わりにタイプ8 (Plain) のウィンドウを開きます:
  - 最後に開いたウィンドウが非表示またはフォームを表示していない、
  - Windows上。
- シートウィンドウはフォームの上に表示されなければならないため、その表示は、ウィンドウにロードされた最初のフォームのOn Loadイベントまで遅延されます (Open window コマンドの例題4参照)。
- 利用シーン: Mac OS Xで、DIALOG, ADD RECORD(...;...*) または同等機能, (Windowsでは非標準)。

Pop up window (32)

このタイプのウィンドウは基本的に Plain dialog box (2) のウィンドウと同じ特徴と機能を持ちますが以下の長所があります:

- 以下の場合ウィンドウは自動で閉じられ、キャンセルイベントがウィンドウに渡されます:
  - ウィンドウの外でクリックされた;
  - 後ろにあるウィンドウまたはMDI (Multiple Document Interface) ウィンドウが移動された;
  - ユーザが**Esc**キーを押した。
- このウィンドウは親ウィンドウの前に表示されます (プロセスのメインウィンドウとして使用してはいけません)。後ろのウィンドウは無効にされていませんが、イベントは受け取りません。
- マウスを使用してウィンドウをリサイズしたり移動したりはできません。しかしこれらのアクションをプログラムで行うと、背景項目の再描画が最適化されます。
- 利用シーン: このタイプのウィンドウの主に3Dベベルやツールバータイプのボタンでポップアップメニューを処理するために使用されます。

Texture appearance (2048)



Mac OS上で、ウィンドウにテクスチャアピアランスを適用することができます。このタイプのルックはMacintoshのあらゆるインターフェースで見ることができます。Windows上ではこのプロパティの効果はありません。

Open window コマンドで作成されたウィンドウにテクスチャアピアランスを適用するには、type 引数に設定するウィンドウタイプにTexture appearance定数を加算します。例えば以下のようにします:

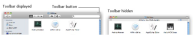
```
$win:=Open window(10;80;-1;-1;Plain window+Texture appearance;"")
```

このルックは以下のタイプのウィンドウに割り当てることができます:

Plain window
Plain no zoom box window
Plain fixed size window
Movable dialog box
Round corner window

Has toolbar button Mac OS (8192)

Mac OSにおいて、4D v12はツールバー管理ボタンを表示できます。この標準ボタンを使用してウィンドウのツールバーを表示したり隠したりできます:



Open window コマンドで開いたウィンドウにツールバー管理ボタンを表示するには、Has toolbar button Mac OS 定数を加算します。例えば以下のようにします:

```
$NewWin:=Open window(10;10;1010;810;Plain window+Has toolbar button Mac OS)
```

ウィンドウのツールバー管理ボタンがユーザーによりクリックされると、On Mac toolbar buttonフォームイベントが生成されます。もちろんプロパティリストにおいて対応するフォームイベントがチェックされていなければなりません。
イベントが生成されるだけです。4Dはウィンドウ中で他の動作を一切行いません。ウィンドウのサイズを調整し、インターフェース要素を表示するのはデベロッパーの作業です。

Has full screen mode Mac (65536)

OS X 環境下の4D v14 では、ドキュメントタイプのウィンドウにおいて新しいオプション「フルスクリーンモード」が追加されています。これを使用すると、ウィンドウの右上隅に「全画面ボタン」が追加されます:

ユーザーがこのアイコンをクリックすると、ウィンドウはフルスクリーンモードになり、メインツールバーが自動的に隠されます。

このオプションを使用するには Has full screen mode Mac という定数を Open window コマンド、Open form window コマンド、Open external window コマンドにて type 引数に渡します。例えば以下のコードは、OS X環境下でフルスクリーンボタンを持ったフォームウィンドウを作成します:

```
$win:=Open form window([Interface];"User_Choice";Plain form window+Has full screen mode Mac)
DIALOG([Interface];"User_Choice")
```

注: Windows 環境下ではこのオプションは何もしません。

CLOSE WINDOW

CLOSE WINDOW {(window)}

引数	型	説明
window	WinRef	⇒ ウィンドウ参照番号, または 省略した場合、カレントプロセスの最前面ウィンドウ

説明

CLOSE WINDOW はカレントプロセスでOpen window や Open form window コマンドで開かれたアクティブウィンドウを閉じます。カスタムウィンドウが開かれていない場合、CLOSE WINDOWは効果を持ちません (システムウィンドウは閉じられません)。CLOSE WINDOW はウィンドウ中でフォームがアクティブであるときに呼び出されても効力を持ちません。CLOSE WINDOWはOpen window や Open form window で開いたウィンドウの利用が終了したときに呼び出します。

Open windowやOpen form windowで事前に開いたウィンドウを閉じる際には、CLOSE WINDOWにウィンドウ参照番号を渡す必要はありません。CLOSE WINDOWは常にこれらのコマンドで開いた最後のウィンドウを閉じます。

WinRefには外部ウィンドウ参照番号を渡すことができます。この引数を渡すと指定した外部ウィンドウが閉じられます。外部ウィンドウに関する詳細はOpen external windowを参照してください。

例題

以下の例題はウィンドウを開き、ADD RECORD コマンドでレコードを追加します。処理が終了したらCLOSE WINDOWでウィンドウを閉じます:

```
Open window(5;40;250;300;0;"New Employee")
Repeat
   ADD RECORD([Employees])  `新規従業員レコードを追加
Until(OK=0)  ` キャンセルされるまでループ
CLOSE WINDOW  ` ウィンドウを閉じる
```

Current form window

Current form window -> 戻り値

引数	型		説明
戻り値	WinRef	↩	カレントフォームウィンドウ参照番号

説明

Current form window コマンドはカレントフォームウィンドウの参照を返します。カレントフォームにウィンドウが設定されていない場合、コマンドは0を返します。

カレントフォームウィンドウはADD RECORDのようなコマンドを使用すると自動で生成されることがあります。またOpen window や Open form window コマンドでも生成されます。

DRAG WINDOW

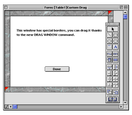
DRAG WINDOW
このコマンドは引数を必要としません

説明

DRAG WINDOW コマンドは、ユーザがマウスのクリックと移動を行うと、ウィンドウをドラッグします。通常このコマンドは (非表示ボタンなど) マウスクリックに瞬時に反応するオブジェクトのオブジェクトメソッドから呼び出します。

例題

以下のフォームには、それぞれの側に非表示ボタンが上に置かれたスタティックピクチャによるフレームがあります:



This window has special borders, you can drag it thanks to the new DRAG WINDOW command.

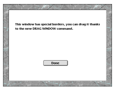
それぞれのボタンには以下のメソッドが書かれています:

```
DRAG WINDOW ` Start dragging window when clicked
```

以下のプロジェクトメソッドを実行後、:

```
Open window(50;50;50+400;50+300;2)
DIALOG([Table1];"Custom Drag")
CLOSE WINDOW
```

以下のようなウィンドウが表示されます:



This window has special borders, you can drag it thanks to the new DRAG WINDOW command.

フレームをクリックして、ウィンドウをドラッグすることができます。

ERASE WINDOW

ERASE WINDOW {(window)}

引数	型		説明
window	WinRef	→	ウィンドウ参照番号, または 省略時、カレントプロセスの最前面ウィンドウ

説明

ERASE WINDOW コマンドはwindowで指定されたウィンドウの内容をクリアします。

window 引数を省略すると、ERASE WINDOWはカレントプロセスの最前面ウィンドウの内容をクリアします。

通常ERASE WINDOWはMESSAGEとGOTO XYと共に使用します。この場合、ERASE WINDOWはウィンドウの内容をクリアし、ウィンドウの左上 (GOTO XY (0; 0)) にカーソルを移動します。

ERASE WINDOWはウィンドウの内容をクリアします。スクリーンからウィンドウを取り除くCLOSE WINDOWと混同しないようにしてください。

Find window

Find window (left ; top {; windowPart}) -> 戻り値

引数	型		説明
left	倍長整数	→	グローバル左座標
top	倍長整数	→	グローバル上座標
windowPart	倍長整数	←	ウィンドウパーツID番号
戻り値	WinRef	↩	ウィンドウ参照番号

説明

Find window コマンドは、left と topに渡した座標のポイントにある最初のウィンドウを返します。

座標は、アプリケーションウィンドウの内容エリア (Windows) 、あるいはメインスクリーン (Macintosh) の左上角からの相対位置で表します。

windowPart 引数を指定すると、ウィンドウが見つかったかどうかにかかわらず、引数は以下の値を返します:

定数	型	値	コメント
In contents	倍長整数	3	Platform: Mac OS and Windows
In drag	倍長整数	4	Platform: Mac OS
In go away	倍長整数	6	Platform: Mac OS
In grow	倍長整数	5	Platform: Mac OS
In menu bar	倍長整数	1	Platform: Mac OS
In system window	倍長整数	2	Platform: Mac OS
In zoom box	倍長整数	8	Platform: Mac OS

Frontmost window

Frontmost window {(*)} -> 戻り値

引数	型	説明
*	演算子 →	指定時, フローティングウィンドウを考慮する 省略時, フローティングウィンドウを無視
戻り値	WinRef ←	ウィンドウ参照番号

説明

Frontmost window コマンドは最前面のウィンドウの参照番号を返します。

GET WINDOW RECT

GET WINDOW RECT (left ; top ; right ; bottom {; window})

引数	型		説明
left	倍長整数	←	ウィンドウの内容領域の左座標
top	倍長整数	←	ウィンドウの内容領域の上座標
right	倍長整数	←	ウィンドウの内容領域の右座標
bottom	倍長整数	←	ウィンドウの内容領域の下座標
window	WinRef	→	ウィンドウ参照番号, または 省略時はカレントプロセスの最前面ウィンドウ, または -1のときはMDIウィンドウ (Windows)

説明

GET WINDOW RECT コマンドはwindowに渡された参照番号を持つウィンドウの座標を返します。ウィンドウが存在しない場合、変数引数は変更されません。

window 引数を省略すると、GET WINDOW RECTはカレントプロセスの最前面ウィンドウを適用します。

座標はアプリケーションウィンドウ (Windows) またはメインスクリーン (Macintosh) の左上隅からの相対座標で表されます。ウィンドウの内容領域 (タイトルバーや枠線は含まれない) の四角座標が返されます。

Note: Windowsではwindowに-1を渡すと、GET WINDOW RECTはアプリケーションウィンドウ (MDI window) の座標を返します。これらの座標はウィンドウの内容領域 (タイトルバーや枠線は含まれない) に対応します。

例題

WINDOW LISTコマンドの例題参照

Get window title

Get window title {(window)} -> 戻り値

引数	型		説明
window	WinRef	→	ウィンドウ参照番号, または 省略時はカレントプロセスの最前面ウィンドウ
戻り値	文字	↩	ウィンドウタイトル

説明

Get window title コマンドは、windowに渡されたウィンドウ参照番号を持つウィンドウのタイトルを返します。ウィンドウが存在しない場合、空の文字列が返されます。

window 引数を省略すると、Get window titleはカレントプロセスの最前面ウィンドウのタイトルを返します。

例題

SET WINDOW TITLEの例題参照

HIDE WINDOW

HIDE WINDOW {(window)}

引数	型		説明
window	WinRef	→	ウィンドウ参照番号、または 省略時、カレントプロセスの最前面ウィンドウ

説明

HIDE WINDOWコマンドは、windowで指定したウィンドウ参照番号を持つウィンドウまたはこの引数省略時はカレントプロセスの最前面ウィンドウを、隠すために使用します。例えばこのコマンドを使用して、複数のプロセスで構成されるアプリケーションで、アクティブなプロセスのウィンドウだけを表示できます。

ウィンドウはスクリーンから消えますが、開かれたままです。プログラムを使用して、4Dのウィンドウがサポートする変更を適用できます。

HIDE WINDOWコマンドで隠したウィンドウを表示するには:

- SHOW WINDOW コマンドにウィンドウ参照番号を渡して使用する。
- ランタイムエクスプローラの**プロセス**ページで、ウィンドウを処理しているプロセスを選択し、**表示**ボタンをクリックする。

プロセスのすべてのウィンドウを隠すにはHIDE PROCESS コマンドを使用します。

例題

この例題は、入力フォームのボタンメソッドです。このメソッドは同じプロセス内で新しいウィンドウにダイアログボックスを開きます。その間入力フォームとツールパレットは隠されます。ダイアログボックスが閉じられると、他のウィンドウは再び表示されます。

```
  ` Object method for the "Information" button

 HIDE WINDOW(Entry) ` Hide the entry window
 HIDE WINDOW(Palette) ` Hide the palette
 $Infos:=Open window(20;100;500;400;8) ` Create the information window
 ... ` Place here instructions that are dedicated to the dialog management
 CLOSE WINDOW($Infos) ` Close the dialog
 SHOW WINDOW(Entry)
 SHOW WINDOW(Palette) ` Display the other windows
```

MAXIMIZE WINDOW

MAXIMIZE WINDOW {(window)}

引数	型		説明
window	WinRef	⇒	ウィンドウ参照番号、または省略時は すべてのカレントプロセス最前面ウィンドウ (Windows) またはカレントプロセスの最前面ウィンドウ (Mac OS)

説明

MAXIMIZE WINDOW コマンドは、windowに渡された参照番号のウィンドウを最大化します。この引数が省略されると、同じ効果がカレントプロセスのすべての最前面ウィンドウ (Windows) またはカレントプロセスの最前面ウィンドウ (Mac OS) に適用されます。

このコマンドは4Dアプリケーションウィンドウのズームボックスをクリックするのと同じ効果があります。

Windows

ウィンドウサイズがアプリケーションウィンドウと同じサイズになります。最大化されたウィンドウは最前面ウィンドウになります。window 引数を渡さないと、コマンドはすべてのアプリケーションウィンドウに適用されます。

Windowsのズームボックス

Mac OS

ウィンドウのサイズが、その内容に合わせて増やされます。window 引数を渡さないと、コマンドはカレントプロセスの最前面ウィンドウに適用されます。

Mac OSのズームボックス

Notes:

- このコマンドはズームボックス付きのウィンドウにのみ適用できます。ズームボックスのないウィンドウの場合、コマンドはなにも行いません。詳細はの節を参照してください。
- Mac OSでは、ズームはウィンドウのコンテンツに基づきます。そのためこのコマンドは、例えばフォームメソッドなど、ウィンドウのコンテンツが定義されているコンテキストで呼ばれなければなりません。そうでなければコマンドはなにも行いません。
- ウィンドウが既に最大化されていれば、コマンドはなにも行いません。

MAXIMIZE WINDOW はウィンドウを最大のサイズに設定します。フォームプロパティでウィンドウのサイズが設定されている場合、ウィンドウのサイズはそれらの値に設定されます。

もう一回ズームボックスをクリックしたり、MINIMIZE WINDOW コマンドを呼び出すと、ウィンドウは元のサイズに縮められます。Windowsでは、引数なしでMINIMIZE WINDOWを呼び出すと、すべてのアプリケーションウィンドウが元のサイズになります。

例題

この例題は、フォームが開かれたときに、そのフォームをフルスクリーンサイズにします。以下のコードをフォームメソッドに置きます:

```
  ` In the Form method

 MAXIMIZE WINDOW
```

MINIMIZE WINDOW

MINIMIZE WINDOW {(window)}

引数	型	説明
window	WinRef ⇨	ウィンドウ参照番号、または省略時は すべてのカレントプロセス最前面ウィンドウ (Windows) またはカレントプロセスの最前面ウィンドウ (Mac OS)

説明

MINIMIZE WINDOW コマンドは、windowに渡された参照番号のウィンドウサイズを、最大化される前に戻します。この引数が省略されると、アプリケーションのそれぞれのウィンドウ (Windows) またはカレントプロセスの最前面ウィンドウ (Mac OS) に適用されます。

このコマンドは4Dアプリケーションウィンドウの縮小ボックスをクリックするのと同じ効果があります。

Windows

ウィンドウサイズが、例えば最大化される前の、元のサイズに設定されます。window 引数を渡さないと、コマンドはすべてのアプリケーションウィンドウに適用されます。

Windowsの縮小ボックス

Mac OS

ウィンドウサイズが、例えば最大化される前の、元のサイズに設定されます。window 引数を渡さないと、コマンドはカレントプロセスの最前面ウィンドウに適用されます。

Mac OS縮小/ズームボックス

コマンドが適用されるウィンドウが事前に (手動またはMAXIMIZE WINDOWを使用して) 最大化されていない場合、またはウィンドウがズームボックスを持たない場合、コマンドは効果がありません。詳細はの節を参照してください。

Note: この機能を、最小化と混同しないようにしてください。最小化は以下のボタンをクリックすると、ボタン (Windows) やDock (Mac OS) にウィンドウが最小化されます:

Windows

Mac OS

Next window

Next window (window) -> 戻り値

引数	型		説明
window	WinRef	→	ウィンドウ参照番号
戻り値	WinRef	↩	ウィンドウ参照番号

説明

Next window コマンドは、window に渡したウィンドウの後ろにあるウィンドウの参照番号を、ウィンドウの重なり順に基づき、返します。

Open external window

Open external window (left ; top ; right ; bottom ; type ; title ; plugInArea) -> 戻り値

引数	型	説明
left	倍長整数	→ ウィンドウ内容領域のグローバル左座標
top	倍長整数	→ ウィンドウ内容領域のグローバル上座標
right	倍長整数	→ ウィンドウ内容領域のグローバル右座標
bottom	倍長整数	→ ウィンドウ内容領域のグローバル下座標
type	倍長整数	→ ウィンドウタイプ
title	文字	→ ウィンドウのタイトル
plugInArea	文字	→ 外部エリアコマンド
戻り値	WinRef	↩ ウィンドウや外部エリアの参照番号

説明

Open external windowは新しいウィンドウを開き、4Dプラグインで提供されるplugInAreaコマンドでサポートされる外部エリアを開きます。plugInAreaに渡されるコードは通常 "_PluginName" の形式です。例: _4D Write, _4D View または _4D Draw。

Open external windowは、(ウィンドウテーマの他のコマンドで使用できる) ウィンドウ参照およびウィンドウに表示された (4Dプラグインが提供する他のルーチンで使用できる) 外部エリアの参照として使用できる、倍長整数値を返します。

最初の6つの引数はOpen window コマンドと同じです。しかしすべての引数が必須です。

Open external window はモードレスウィンドウを作成します。コマンドはユーザの入力を待ちませんので、複数のアクティブなウィンドウを一度に起動できます。ウィンドウをクリックで切り替え、最前面のウィンドウを編集できます。ウィンドウがタイトルバーを持つ場合、コントトールメニューボックス (Windows) あるいはクローズボックス (Macintosh) が追加され、ユーザはこのボックスをクリックしてウィンドウを閉じることができます。

例題

以下の例題は外部ウィンドウを開き、4D Write外部エリアを表示します:

```
wrWind:=Open external window(50;50;350;450;8;"Letter Writing";"_4D WRITE")
```

以下の例題は、前に開いた外部ウィンドウを閉じます:

```
CLOSE WINDOW(wrWind)
```

Open form window

Open form window ({aTable ;} formName {; type {; hPos {; vPos {; *}}}}) -> 戻り値

引数	型		説明
aTable	テーブル	⇒	フォームが属するテーブル、または省略時デフォルトテーブル
formName	文字	⇒	フォーム名
type	倍長整数	⇒	ウィンドウタイプ
hPos	倍長整数	⇒	ウィンドウの横位置
vPos	倍長整数	⇒	ウィンドウの縦位置
*	演算子	⇒	ウィンドウの現在の位置とサイズを保存
戻り値	WinRef	⇐	ウィンドウ参照番号

説明

Open form window コマンドはフォームformNameのサイズとリサイズプロパティを使用して、新しいウィンドウを開きます。

formNameフォームはウィンドウに表示されません。フォームを表示するには、フォームをロードするコマンド (ADD RECORD 等) を呼び出さなければなりません。

デフォルトで (type 引数が渡されないと)、クローズボックス付きの標準ウィンドウが開かれます。Open window コマンドと異なり、クローズボックスにはメソッドは割り当てられません。クローズボックスをクリックすると、On Close Boxフォームイベントが有効にされている場合を除き、ウィンドウがキャンセルされ閉じられます。On Close Boxフォームイベントが有効であれば、割り当てられたコードが実行されます。

formNameフォームがリサイズ可能であれば、開かれるウィンドウにはズームボックスとグローボックスが付加されます。

Note: フォームの主なプロパティを取得するには FORM GET PROPERTIES コマンドを使用します。

任意のtype引数は、ウィンドウのタイプを指定するために使用します。以下のいずれかの定数を渡さなければなりません (Open Form Window テーマ内):

定数	型	値
Has full screen mode Mac	倍長整数	65536
Has toolbar button Mac OS	倍長整数	8192
Modal form dialog box	倍長整数	1
Movable form dialog box	倍長整数	5
Palette form window	倍長整数	1984
Plain form window	倍長整数	8
Pop up form window	倍長整数	32
Sheet form window	倍長整数	33

Notes:

- (グローボックス、クローズボックスなど...) 作成されたウィンドウの属性は、選択されたtypeに対するOSのインタフェース仕様に基づきます。ゆえに使用するプラットフォームによって異なる結果が得られる場合があります。
- Has toolbar button Mac OS と Has full screen mode Mac 定数は、他の定数に加算して使用しなければなりません。
- ウィンドウタイプに関する詳細は ウィンドウタイプ のセクションを参照してください。ただし Open Form Window テーマの定数のみが Open form window コマンドで利用可能であることに注意してください。

オプションの引数 hPos を使用して、ウィンドウの横位置を指定できます。特定の位置をピクセル単位で指定するか (Open window コマンド参照)、Open Form Window テーマの以下の定義済み定数を渡します:

定数	型	値
Horizontally Centered	倍長整数	65536
On the Left	倍長整数	131072
On the Right	倍長整数	196608

オプションの引数 vPos を使用して、ウィンドウの縦位置を指定できます。特定の位置をピクセル単位で指定するか (Open window コマンド参照)、Open Form Window テーマの以下の定義済み定数を渡します:

定数	型	値
At the Bottom	倍長整数	393216
At the Top	倍長整数	327680
Vertically Centered	倍長整数	262144

これらの引数はツールバーとメニューバーの表示状態、およびWindowsではアプリケーションウィンドウ現在のサイズを考慮します。

オプションの引数 * を渡すと、閉じられるときにその時点での位置とサイズが記憶されます。ウィンドウが再度開かれると、以前の位置とサイズが再現されます。この場合、vPos と hPos 引数はウィンドウが最初に開かれるときにのみ使用されます。

例題 1

以下のコードはクローズボックス付きの標準のウィンドウを開き、自動で"Input"フォームのサイズに調整します。フォームはリサイズ可能に設定されているので、ウィンドウはグローとズームボックスを持ちます:

```
$winRef :=Open form window([Table1];"Enter")
```

例題 2

以下のコードはプロジェクトフォーム"Tools"に基づき、スクリーンの左上の位置にフローティングパレットを開きます。このパレットは閉じられた時の位置を記憶し、再度開かれるときにはその位置が使用されます:

```
$winRef :=Open form window("Tools";Palette form window;On the Left;At the Top;*)
```

Open window

Open window (left ; top ; right ; bottom {; type {; title {; controlMenuBox}}}) -> 戻り値

引数	型		説明
left	倍長整数	→	ウィンドウ内容領域のグローバル左座標
top	倍長整数	→	ウィンドウ内容領域のグローバル上座標
right	倍長整数	→	ウィンドウ内容領域のグローバル右座標, または-1でフォームのデフォルトサイズを使用
bottom	倍長整数	→	ウィンドウ内容領域のグローバル下座標, または-1でフォームのデフォルトサイズを使用
type	倍長整数	→	ウィンドウタイプ
title	文字	→	ウィンドウのタイトル または "" でデフォルトのフォームタイトルを使用
controlMenuBox	文字	→	コントロールメニューボックスがダブルクリック またはクローズボックスがクリックされたときに 呼び出すメソッド
戻り値	WinRef	←	ウィンドウ参照番号

説明

Open windowは最初の4つの引数で指定された寸法を使用して新しいウィンドウを開きます。

- left はアプリケーションウィンドウの左端から、ウィンドウ内側の左端までの距離 (ピクセル単位) です。
- top はアプリケーションウィンドウの上端から、ウィンドウ内側の上端までの距離 (ピクセル単位) です。
- right はアプリケーションウィンドウの左端から、ウィンドウ内側の右端までの距離 (ピクセル単位) です。
- bottom はアプリケーションウィンドウの上端から、ウィンドウ内側の下端までの距離 (ピクセル単位) です。

right と bottom両方に -1 を渡すと、4Dは以下の条件のときウィンドウのサイズを自動で決定します:

- デザインモードのフォームエディタで、フォームプロパティのサイズオプションを設定している
- Open windowを呼び出す前に、FORM SET INPUTコマンドで、* 引数付きでフォームを選択している。

重要: この自動サイズ決定は、事前に表示するフォームをFORM SET INPUTコマンドで指定し、またオプションの * 引数をFORM SET INPUTコマンドに指定したときにのみ、実行されます。

- type 引数はオプションで、で説明している異なるウィンドウタイプから、表示したいウィンドウを選択するために使用します。ウィンドウタイプが負数の時、フローティングウィンドウが作成されます。typeが指定されない時、タイプ1がデフォルトで使用されます。
- title 引数はオプションで、ウィンドウのタイトルを指定します。

空の文字列 ("") をtitleに渡すと、4Dはデザインモードのフォームエディタで、フォームプロパティに指定されたウィンドウのタイトルを使用します。

重要: デフォルトのフォームタイトルは表示するフォームを事前にFORM SET INPUTで指定し、かつFORM SET INPUTにオプションの * 引数を渡したときにのみ設定されます。

- controlMenuBox 引数はオプションで、ウィンドウのコントロールボックスメソッドを指定します。この引数が指定されると、コントロールメニューボックス (Windows) やクローズボックス (Macintosh) がウィンドウに追加されます。ユーザがコントロールメニューボックスをダブルクリック (Windows) またはクローズボックスをクリック (Macintosh) すると、controlMenuBox に渡したメソッドが呼び出されます。

Note: On Close Boxイベントを使用して、フォームメソッドでウィンドウを閉じる際のコントロールを行うこともできます。詳細はForm eventコマンドを参照してください。

プロセス内で複数のウィンドウが開かれている場合、最後に開かれたウィンドウがそのプロセス内でアクティブ (最前面) です。アクティブウィンドウ内の情報のみが更新可能です。他のウィンドウは見ることはできます。

フォームは開かれたウィンドウ内に表示されます。MESSAGE コマンドのテキストもウィンドウに表示されます。

例題 1

以下のプロジェクトメソッドはメインウィンドウ (Windows) またはメインスクリーン (Macintosh) の中央にウィンドウを開きます。2~4つの引数を受け入れます:

```
  ` OPEN CENTERED WINDOW project method
  ` $1 – ウィンドウ幅
  ` $2 – ウィンドウ高さ
  ` $3 – ウィンドウタイプ （オプション）
  ` $4 – ウィンドウタイトル （オプション）
 $SW:=Screen width\2
 $SH:=(Screen height\2)
 $WW:=$1\2
 $WH:=$2\2
 Case of
    :(Count parameters=2)
       Open window($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH)
    :(Count parameters=3)
       Open window($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH;$3)
    :(Count parameters=4)
       Open window($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH;$3;$4)
 End case
```

プロジェクトメソッド記述後、以下のように使用できます:

```
 OPEN CENTERED WINDOW(400;250;Movable dialog box;"Update Archives")
 DIALOG([Utility Table];"UPDATE OPTIONS")
 CLOSE WINDOW
 If(OK=1)
  ` ...
 End if
```

例題 2

以下の例題はコントロールメニューボックス (Windows) またはクローズボックス (Macintosh) メソッド付きのウィンドウを開きます。ウィンドウはアプリケーションウィンドウの右上に開かれます。

```
 Open window(Screen width-149;33;Screen width-4;178;-Palette window;"";"CloseColorPalette")
 DIALOG([Dialogs];"Color Palette")
```

CloseColorPalette メソッドは CANCEL コマンドを呼び出します:

```
 CANCEL
```

例題 3

以下の例題では、ウィンドウに表示するフォームのサイズとタイトルプロパティを使用して、ウィンドウを開きます:

```
 FORM SET INPUT([Customers];"Add Records";*)
 Open window(10;80;-1;-1;Plain window;"")
 Repeat
    ADD RECORD([Customers])
 Until(OK=0)
```

Reminder: Open windowが自動でフォームのプロパティを使用するためには、事前に表示するフォームをFORM SET INPUTコマンドで指定し、またオプションの * 引数をFORM SET INPUTコマンドに指定しなければなりません。またデザインモードで対応するフォームのプロパティを設定していなければなりません。

例題 4

この例題はMac OS X下でのシートウィンドウを表示する際の遅延メカニズムを説明します:

```
 $myWindow:=Open window(10;10;400;400;Sheet window)
  ` 当面、ウィンドウは作成されるが隠されている
 DIALOG([table];"dialForm")
  `On Loadイベントが生成され、シートウィンドウが表示される。
  `ウィンドウはタイトルバーからドロップダウンされる
```

REDRAW WINDOW

REDRAW WINDOW {(window)}

引数	型		説明
window	WinRef	→	ウィンドウ参照番号, または省略時 カレントプロセスの最前面ウィンドウ

説明

REDRAW WINDOW コマンドはwindowに渡した参照番号を持つウィンドウのグラフィックな更新を行います。

window 引数を省略すると、REDRAW WINDOWはカレントプロセスの最前面ウィンドウに適用されます。

Note: 4Dはウィンドウが移動、リサイズ、最前面に移動、フォームの変更、またウィンドウ中の値が変更されるたびにグラフィックの更新を処理します。このコマンドを使用することはほとんどありません。

RESIZE FORM WINDOW

RESIZE FORM WINDOW (width ; height)

引数	型		説明
width	倍長整数	⇒	現在のフォームウィンドウ幅に追加あるいは取り除くピクセル数
height	倍長整数	⇒	現在のフォームウィンドウ高に追加あるいは取り除くピクセル数

説明

RESIZE FORM WINDOW コマンドはカレントフォームウィンドウのサイズを変更します。

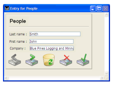
width と height引数には、現在のウィンドウサイズに追加したいピクセル数を渡します。現在のサイズを変更したくない場合は0を渡します。サイズを小さくするには、width と height に負数を渡します。

このコマンドはリサイズボックスを使用した手動によるサイズ変更と同じ効果を生み出します (ウィンドウのタイプがサイズ変更を許可する場合)。結果、コマンドはフォームプロパティで定義されたサイズの制限やサイズ変更のオブジェクトプロパティを考慮します。例えばフォームが許可するサイズよりも大きなサイズにウィンドウをリサイズしようとした場合、コマンドの効果はありません。

このコマンドの動作はSET WINDOW RECT コマンドの動作と異なることに注意してください。SET WINDOW RECT コマンドはウィンドウのリサイズ時にフォームのプロパティやコンテンツを考慮に入れません。またこのコマンドはフォームサイズを変更する必要がないことにも留意してください。フォームのサイズをプログラムで変更するにはFORM SET SIZE コマンドを参照してください。

例題

以下のウィンドウがあります (フィールドとフレームに水平方向に拡大のプロパティが設定されています):



Entry for People

People

Last name : Smith

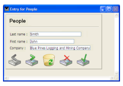
First name : John

Company : Blue Pines Logging and Minin

以下の行を実行すると:

```
RESIZE FORM WINDOW(25;0)
```

ウィンドウは以下のように表示されます:



Entry for People

People

Last name : Smith

First name : John

Company : Blue Pines Logging and Mining Company

SET WINDOW RECT

SET WINDOW RECT (left ; top ; right ; bottom {; window}{; *})

引数	型		説明
left	倍長整数	⇒	ウィンドウ内容領域のグローバル左座標
top	倍長整数	⇒	ウィンドウ内容領域のグローバル上座標
right	倍長整数	⇒	ウィンドウ内容領域のグローバル右座標
bottom	倍長整数	⇒	ウィンドウ内容領域のグローバル下座標
window	WinRef	⇒	ウィンドウ参照番号, または省略時カレントプロセスの最前面ウィンドウ
*	演算子	⇒	省略時 (デフォルト) = ウィンドウを最前面にする 指定時 = ウィンドウの並びレベルを変更しない

説明

SET WINDOW RECTコマンドはwindowに渡した参照番号のウィンドウのグローバル座標を変更します。ウィンドウが存在しない場合、コマンドはなにも行いません。

window 引数を省略すると、**SET WINDOW RECT**はカレントプロセスの最前面ウィンドウに適用されます。

このコマンドは、渡された新座標に基づき、ウィンドウをリサイズして移動します。

座標は、アプリケーションウィンドウの内容領域 (Windows) またはメインスクリーン (Macintosh) の左上隅に対する相対座標で表します。座標はウィンドウの内容領域に対応する四角を表します (タイトルバーと枠線を除く)。

警告: このコマンドを使用すると、ウィンドウをメインウィンドウ (Windows) やスクリーン (Macintosh) の外に動かすことができてしまうことに注意してください。これを避けるにはScreen widthやScreen heightなどのコマンドを使用してウィンドウの新しい座標を検証します。

このコマンドを実行するとデフォルトで自動的に、window 引数で指定されたウィンドウが最前面に移動されます (window引数が使用されている場合)。最後の引数として * を渡すとこの動作を無効にできます。この場合コマンドはウィンドウの元の順番 (z座標) を変更しません。

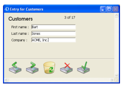
このコマンドはフォームオブジェクトには影響しません。ウィンドウに表示されているフォーム中のフォームオブジェクトは、(プロパティ設定にかかわらず) このコマンドにより移動やリサイズはされません。ウィンドウのみが更新されます。オブジェクトのリサイズプロパティを考慮に入れたフォームウィンドウのサイズ変更を行うにはRESIZE FORM WINDOWコマンドを使用します。

例題 1

WINDOW LISTコマンドの例題参照

例題 2

以下のウィンドウがあるとき:

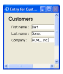


First name : Bart
Last name : Jones
Company : ACME, Inc.

以下のコードを実行すると:

```
SET WINDOW RECT(100;100;300;300)
```

以下のように表示されます:



First name : Bart
Last name : Jones
Company : ACME, Inc.

SET WINDOW TITLE

SET WINDOW TITLE (title {; window})

引数	型		説明
title	文字	⇨	ウィンドウタイトル
window	WinRef	⇨	ウィンドウ参照番号, または 省略時カレントプロセスの最前面ウィンドウ

説明

SET WINDOW TITLE コマンドはwindow に渡した参照番号のウィンドウのタイトルを、titleに渡したテキストに変更します (最大80文字)。

ウィンドウが存在しない場合、SET WINDOW TITLEは何も行いません。

window 引数を省略すると、SET WINDOW TITLEはカレントプロセスの最前面ウィンドウのタイトルを変更します。

Note: デザインモードで4Dはウィンドウタイトルを自動で変更します。例えばデータ入力時には"更新：テーブル"となります。ウィンドウタイトルを変更しても、4Dがそれを上書きしてしまいます。他方、アプリケーションモードでは4Dはウィンドウタイトルの変更を行いません。

例題

フォームでデータ入力を行っている間、長い時間のかかる処理を行うためにボタンをクリックします (例えばプログラムでリレートするレコードをブラウズするなど)。カレントウィンドウのタイトルを使用して進捗状況を知らせることができます:

```
  ` bAnalysis button Object Method
 Case of
    :(Form event=On Clicked)
  ` 現在のタイトルを保持
       $vsCurTitle:=Get window title
  ` 時間のかかる処理を開始
       FIRST RECORD([Invoice Line Items])
       For($vlRecord;1;Records in selection([Invoice Line Items]))
          DO SOMETHING
  ` 進捗状況を表示
          SET WINDOW TITLE("Processing Line Item #"+String($vlRecord))
       End for
  ` 元のウィンドウタイトルに戻す
       SET WINDOW TITLE($vsCurTitle)
 End case
```

SHOW WINDOW

SHOW WINDOW {(window)}

引数	型	説明
window	WinRef	→ ウィンドウ参照番号または 省略時カレントプロセスの最前面ウィンドウ

説明

SHOW WINDOW コマンドはwindowに渡した参照番号のウィンドウを表示します。この引数が省略されていると、カレントプロセスの最前面ウィンドウが表示されます。

SHOW WINDOWコマンドを使用するには、ウィンドウがHIDE WINDOW コマンドを使用して隠されていなければなりません。ウィンドウがすでに表示されていればコマンドはなにも行いません。

例題

HIDE WINDOW コマンドの例題参照

Window kind

Window kind {(window)} -> 戻り値

引数	型		説明
window	WinRef	→	ウィンドウ参照番号, または 省略時カレントプロセスの最前面ウィンドウ
戻り値	倍長整数	↩	ウィンドウのタイプ

説明

Window kind コマンドはwindowに渡した参照番号のウィンドウのタイプを返します。ウィンドウが存在しない場合、Window kind には 0 が返されます。

そうでなければWindow kind は以下の値を返します:

定数	型	値
External window	倍長整数	5
Floating window	倍長整数	14
Modal dialog	倍長整数	9
Regular window	倍長整数	8

window 引数を省略すると、Window kind はカレントプロセスの最前面ウィンドウのタイプを返します。

例題

WINDOW LISTコマンドの例題参照

WINDOW LIST

WINDOW LIST (windows {; *})

引数	型		説明
windows	配列	←	ウィンドウ参照番号の配列
*	演算子	⇒	指定時, フローティングウィンドウも含める 省略時, フローティングウィンドウを含めない

説明

WINDOW LIST コマンドは配列 windows を生成し、実行中のすべての (カーネルおよびユーザ) プロセスで開かれているウィンドウの参照番号を返します。

オプションの * 引数を渡さない場合、フローティングウィンドウは含められません。

例題

以下のプロジェクトメソッドは、(フローティングとダイアリグボックスを除く) すべての開かれているウィンドウをタイル表示します:

```
  ` TILE WINDOWS project method

 WINDOW LIST($alWnd)
 $vlLeft:=10
 $vlTop:=80  ` ツールバーのスペースを確保
 For($vlWnd;1;Size of array($alWnd))
    If(Window kind($alWnd{$vlWnd})#Modal dialog)
       GET WINDOW RECT($vlWL;$vlWT;$vlWR;$vlWB;$alWnd{$vlWnd})
       $vlWR:=$vlLeft+($vlWR-$vlWL)
       $vlWB:=$vlTop+($vlWB-$vlWT)
       $vlWL:=$vlLeft
       $vlWT:=$vlTop
       SET WINDOW RECT($vlWL;$vlWT;$vlWR;$vlWB;$alWnd{$vlWnd})
       $vlLeft:=$vlLeft+10
       $vlTop:=$vlTop+25
    End if
 End for
```

Note: このメソッドにメインウィンドウ (Windows) やスクリーン (Mac OS) のテストを組み込むとより洗練されたものとなります。

Window process

Window process {(window)} -> 戻り値

引数	型		説明
window	WinRef	→	ウィンドウ参照番号
戻り値	倍長整数	↩	プロセス参照番号

説明

Window process コマンドはwindowに渡されたウィンドウが実行されているプロセスの番号を返します。ウィンドウが存在しない場合0が返されます。

window 引数を省略すると、Window processは現在の最前面ウィンドウのプロセスを返します。

オブジェクト(フォーム)

- オブジェクトプロパティ
- GET STYLE SHEET INFO New 14.0
- LIST OF STYLE SHEETS New 14.0
- OBJECT DUPLICATE
- OBJECT Get action New 14.0
- OBJECT Get auto spellcheck
- OBJECT GET BEST SIZE
- OBJECT Get border style New 14.0
- OBJECT Get context menu New 14.0
- OBJECT GET COORDINATES
- OBJECT Get data source New 14.0
- OBJECT GET DRAG AND DROP OPTIONS
- OBJECT Get enabled
- OBJECT Get enterable
- OBJECT GET EVENTS New 14.0
- OBJECT Get filter
- OBJECT Get focus rectangle invisible
- OBJECT Get font
- OBJECT Get font size
- OBJECT Get font style
- OBJECT Get format
- OBJECT Get help tip
- OBJECT Get horizontal alignment
- OBJECT Get indicator type New 14.0
- OBJECT Get keyboard layout
- OBJECT Get list name Updated 14.0
- OBJECT Get list reference New 14.0
- OBJECT GET MAXIMUM VALUE New 14.0
- OBJECT GET MINIMUM VALUE New 14.0
- OBJECT Get multiline New 14.0
- OBJECT Get placeholder New 14.0
- OBJECT GET PRINT VARIABLE FRAME New 14.0
- OBJECT GET RESIZING OPTIONS
- OBJECT GET RGB COLORS
- OBJECT GET SCROLL POSITION
- OBJECT GET SCROLLBAR
- OBJECT GET SHORTCUT
- OBJECT Get style sheet New 14.0
- OBJECT GET SUBFORM
- OBJECT GET SUBFORM CONTAINER SIZE
- OBJECT Get text orientation New 14.0
- OBJECT Get three states checkbox New 14.0
- OBJECT Get title
- OBJECT Get type New 14.0
- OBJECT Get vertical alignment
- OBJECT Get visible
- OBJECT Is styled text New 14.0
- OBJECT MOVE
- OBJECT SET ACTION New 14.0
- OBJECT SET AUTO SPELLCHECK
- OBJECT SET BORDER STYLE New 14.0
- OBJECT SET COLOR
- OBJECT SET CONTEXT MENU New 14.0
- OBJECT SET COORDINATES New 14.0
- OBJECT SET DATA SOURCE New 14.0
- OBJECT SET DRAG AND DROP OPTIONS
- OBJECT SET ENABLED
- OBJECT SET ENTERABLE
- OBJECT SET EVENTS New 14.0
- OBJECT SET FILTER
- OBJECT SET FOCUS RECTANGLE INVISIBLE
- OBJECT SET FONT Updated 14.0
- OBJECT SET FONT SIZE
- OBJECT SET FONT STYLE
- OBJECT SET FORMAT
- OBJECT SET HELP TIP
- OBJECT SET HORIZONTAL ALIGNMENT
- OBJECT SET INDICATOR TYPE New 14.0
- OBJECT SET KEYBOARD LAYOUT
- OBJECT SET LIST BY NAME Updated 14.0
- OBJECT SET LIST BY REFERENCE New 14.0
- OBJECT SET MAXIMUM VALUE New 14.0
- OBJECT SET MINIMUM VALUE New 14.0
- OBJECT SET MULTILINE New 14.0
- OBJECT SET PLACEHOLDER New 14.0
- OBJECT SET PRINT VARIABLE FRAME New 14.0
- OBJECT SET RESIZING OPTIONS
- OBJECT SET RGB COLORS Updated 14.0
- OBJECT SET SCROLL POSITION
- OBJECT SET SCROLLBAR
- OBJECT SET SHORTCUT
- OBJECT SET STYLE SHEET New 14.0
- OBJECT SET SUBFORM
- OBJECT SET TEXT ORIENTATION New 14.0
- OBJECT SET THREE STATES CHECKBOX New 14.0
- OBJECT SET TITLE
- OBJECT SET VERTICAL ALIGNMENT
- OBJECT SET VISIBLE
- DISABLE BUTTON
- ENABLE BUTTON

オブジェクトプロパティ

オブジェクトプロパティコマンドはフォーム上のオブジェクトのプロパティに作用します。これにより、アプリケーションモードでレコードを表示するフォームを使用する際、オブジェクトの外観と振る舞いを変更することが可能になります。

重要: これらのコマンドの範囲は現在使われているフォームです。フォームを終了すると変更内容は失われます。

オブジェクト名またはデータソース名によるオブジェクトへのアクセス

オブジェクトプロパティコマンドは、以下の汎用シンタックスを共有します:

コマンド名 ({*;} object {; コマンド毎の追加の引数)

オプションの引数 * を指定する場合、objectにオブジェクト名 (文字列) を指定します。

1回の呼び出しでフォーム上の複数のオブジェクトを指定するために、オブジェクト名に@文字を使用できます。以下の表に、コマンド対して指定可能なオブジェクト名の例を示します。

オブジェクト名	**呼び出しの影響を受けるオブジェクト**
mainGroupBox	mainGroupBoxオブジェクトのみ
main@	オブジェクト名が"main"で始まるオブジェクト
@GroupBox	オブジェクト名が"GroupBox"で終わるオブジェクト
@Group@	オブジェクト名が"Group"を含むオブジェクト
main@Btn	オブジェクト名が"main"で始まり"Btn"で終わるオブジェクト
@	フォーム上のすべてのオブジェクト

注: @文字が文字列の中に含まれていた時の処理の仕方を設定することができます。このオプションは"オブジェクト (フォーム)"テーマ内のコマンドの機能に影響します。詳細な情報に関しては、4D *Design Reference* マニュアルを参照して下さい。

オプションの引数 * を省略する場合、object引数にはフィールドや変数を指定します。この場合文字列ではなく、フィールドまたは変数参照（フィールドまたは変数オブジェクトのみ）を指定します。

汎用コマンドとマルチスタイルエリアの関係性

4D v14 以降、OBJECT SET RGB COLORS や OBJECT SET FONT STYLE といった汎用コマンドとマルチスタイルエリアとの関係性が新しくなりました。

以前のバージョンの4Dでは、これらのコマンドのどれかを実行すると、エリア内に挿入されたカスタムのスタイルタグを全て変更してしまいました。v14からは、デフォルトのプロパティ(とデフォルトのタグで保存されたプロパティ)のみが変更されるようになりました。カスタムのスタイルタグはそのままの状態を維持します。

例えば、以下の様なマルチスタイルエリアにデフォルトのタグが保存されていた場合を考えます。

This is the word red

このエリアのプレーンテキストは以下のようになります:

```
<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#009900">This is the word <span
style="color:#D81E05">red</span></span>
```

以下のコードを実行した場合:

```
 OBJECT SET COLOR(*;"myArea";-(Blue+(256*Yellow)))
```

4D v14 では、赤文字の部分はそのまま赤文字として残ります。

4D v14	**以前のバージョン**
 This is the word red	 This is the word red
`<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#0000FF">This is the word <span style="color:#D81E05">red</span></span>`	`<span style="font-family:'Segoe UI';font-size:9pt;text-align:left;font-weight:normal;font-style:normal;text-decoration:none;color:#0000FF;"><span style="background-color:#FFFFFF">This is the word red</span></span>`

これが適用される汎用コマンドは、以下の5つです:

OBJECT SET RGB COLORS
OBJECT SET COLOR
OBJECT SET FONT
OBJECT SET FONT STYLE
OBJECT SET FONT SIZE

マルチスタイルエリアにおいては、汎用コマンドはデフォルトのスタイルを設定するためだけに使用されるべきです。データベースの実行中にスタイルを管理するためには、"Styled Text" テーマ内のコマンドを使用することが推奨されます。

GET STYLE SHEET INFO

GET STYLE SHEET INFO (styleSheetName ; font ; size ; styles)

引数	型		説明
styleSheetName	テキスト	→	スタイルシート名
font	テキスト	←	フォント名
size	倍長整数	←	フォントサイズ
styles	倍長整数	←	書体

説明

GET STYLE SHEET INFOコマンドは、styleSheetNameパラメーターで指定したスタイルシートの現在の構成を返します。

styleSheetName 引数には、デザインモードで定義されたスタイルシート名を渡します。 自動スタイルシートを指定するためには、"Font Styles" テーマ内にある Automatic style sheet 定数を使用して下さい。

定数	型	値
Automatic style sheet	文字列	__automatic__

font 引数には、カレントのプラットフォームのスタイルシートと関連付けされているフォントの名前がコマンドから返されます。

size 引数には、カレントのプラットフォームのスタイルシートと関連付けされている文字サイズがポイント数でコマンドから返されます。

styles 引数には、カレントのプラットフォームのスタイルシートと関連付けされているスタイルに対応した値がコマンドから返されます。返された値に対応するスタイルは、以下の表のとおりで、"Font Styles" テーマ内にあります。

定数	型	値
Bold	倍長整数	1
Bold and Italic	倍長整数	3
Bold and Underline	倍長整数	5
Italic	倍長整数	2
Italic and Underline	倍長整数	6
Plain	倍長整数	0
Underline	倍長整数	4

このコマンドが正しく実行された場合には、*OK* システム変数は 1 に変更されます。そうでない(例えば styleSheetName 引数で指定したスタイルシートが存在しないなどの)場合は、OK システム変数は 0 に設定されます。

例題

自動スタイルシートのカレント設定を調べたい場合:

```
C_LONGINT($size;$style)
C_TEXT($font)
GET STYLE SHEET INFO(Automatic style sheet;$font;$size;$style)
```

LIST OF STYLE SHEETS

LIST OF STYLE SHEETS (arrStyleSheets)

引数	型		説明
arrStyleSheets	テキスト配列	←	アプリケーション内の定義済みスタイルシート名一覧

説明

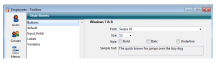
LIST OF STYLE SHEETSコマンドは、arrStyleSheets配列にアプリケーション内のスタイルシートのリストを返します。.

arrStyleSheets テキスト配列が未定義だった場合、この配列はコマンドによって自動的に作成されます。配列のサイズは、定義されているスタイルシートの数に応じて自動的に決められます。

コマンド実行後、配列の各要素にはスタイルシートの名前が格納されます。これらの名前はスタイルシートエディターと同様にアルファベット順にソートされます。配列の第1要素には 自動スタイルシートを意味する "__automatic__"が必ず入ります。

例題

アプリケーション内に、以下のようにスタイルシートが定義されていたとします:


Employees - Toolbox
Users
Groups
Menus
Style Sheets
Buttons
default
Input_fields
Labels
Variables
Windows 7 & 8

Font: Segoe UI
Size: 12
Style: ☐ Bold ☐ Italic ☐ Underline
Sample Text: The quick brown fox jumps over the lazy dog

下のコードを実行すると、戻り値の配列の要素には以下の様に値が格納されています:

```
LIST OF STYLE SHEETS($arrStyles)
 // $arrStyles{1} には "__automatic__"
 // $arrStyles{2} には "Buttons"
 // $arrStyles{3} には "Input_fields"
 // $arrStyles{4} には "Default"
 // $arrStyles{5} には "Labels"
 // $arrStyles{6} には "Variables"
```

OBJECT DUPLICATE

OBJECT DUPLICATE ({* ;} object {; newName {; newVar {; boundTo {; moveH {; moveV {; resizeH {; resizeV}}}}}}} {; *})

引数	型		説明
*	演算子	⇒	指定時: objectはオブジェクト名 (文字列)、省略時: objectは変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名 (*指定時)、または変数やフィールド (*省略時)
newName	テキスト	⇒	新しいオブジェクトの名前
newVar	ポインター	⇒	新しいオブジェクトの変数へのポインター
boundTo	テキスト	⇒	直前の入力順の入力可能オブジェクトまたはラジオボタングループ
moveH	倍長整数	⇒	新しいオブジェクトの横シフト (>0 = 右方向, <0 = 左方向)
moveV	倍長整数	⇒	新しいオブジェクトの縦シフト (>0 = 下方向, <0 = 上方向)
resizeH	倍長整数	⇒	オブジェクトの横リサイズ
resizeV	倍長整数	⇒	新しいオブジェクトの縦リサイズ
*	演算子	⇒	指定時= 絶対座標, 省略時= 相対座標

説明

OBJECT DUPLICATEコマンドを使用して、object引数で指定したオブジェクトのコピーを作成できます。コピーはアプリケーションモードで実行されているフォームのコンテキストで生成されます。デザインモードのソースフォームは変更されません。

デフォルトで、割り当てられているオブジェクトメソッドを含む、ソースオブジェクトに対しプロパティリストで設定されているすべてのオプションがコピーに適用されます (サイズ、リサイズオプション、カラー等)。
しかし以下の例外について留意してください:

- デフォルトボタン: フォーム中にデフォルトボタンは1 つだけ存在できます。" デフォルトボタン" プロパティを持つボタンを複製すると、このプロパティはコピーに割り当てられ、ソースオブジェクトからは取り除かれます。
- キーボードショートカット: ソースオブジェクトに割り当てられているキーボードショートカットは複製されません。このプロパティはコピー先では未設定となります。
- オブジェクト名: フォーム中でオブジェクト名はユニークでなければなりません。newName引数を渡さない場合、ソースオブジェクト名が自動でインクリメントされ新しいオブジェクトで使用されます (後述参照)。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字) です。この引数を渡さないと、object はフィールドまたは変数です。この場合、文字ではなくフィールドまたは変数への参照 (フィールドまたは変数オブジェクトのみ) を渡します。
フィールドや変数参照を渡した場合で、フォーム中に同じ参照を使用するオブジェクトが複数ある場合、最初に見つかったオカレンスが使用されます。このような曖昧さを避けるために、ユニークであるオブジェクト名の使用をお勧めします。

newName 引数にはオブジェクトのコピーに割り当てる名前を渡します。この名前はオブジェクト名の命名規則に沿い、フォーム中でユニークでなければなりません。有効でない、あるいは既に使用されている名前を渡すと、コマンドはなにも行わず、OK 変数に0 が設定されます。
この引数を省略するか空の文字列を渡すと、ソースオブジェクト名をインクリメントすることで、新しい名前が自動生成されます。例えば:

ソース名	コピー名
Button	Button1
Button20	Button21
Button21	Button23 (Button22が既に存在すれば)

newVar には新しいオブジェクトに割り当てられる変数へのポインタを渡します。ルールとして、ソースオブジェクトと同じ型の変数をポイントしなければなりません。しかし特定の種類の" 型変換" が可能です。汎用的なコードを書けるようにするために、コマンドは自動処理を提供します:

- 通常すべての " 入力可" 変数は型変換が可能です。例えば日付や倍長整数を表示するオブジェクトを複製し、テキスト型の変数を割り当てることができます。互換のあるプロパティは保持されます。またテキストオブジェクトとピクチャオブジェクト間の型の変更も許容します。テキストオブジェクトを複製してブール変数やフィールドを割り当てると、自動でチェックボックスとして表示される点に留意してください。
- 通常動的に変数からフィールド、あるいはその逆に変換が可能です。
  他方、グラフィックオブジェクト (ボタン、チェックボックス等) を他のタイプのコントロールに変換することはできません。

変数の型がオブジェクトと互換でない場合、コマンドはなにも行わず、OK変数に0が設定されます。この引数を省略すると、4D が変数を動的に作成します (の"ダイナミック変数" 参照)。スタティックオブジェクト (線、四角、スタティックピクチャ等) を複製するとき、この引数は無視されます。他の引数を使用できるようにするには Nil ポインタ (->[]) を渡します。

boundTo 引数は2 つのケースで使用します:

- 入力順の変更: この場合boundTo には複製したオブジェクトの直前の入力順の入力可オブジェクト名を渡します。新しいオブジェクトをページ中最初の入力順にしたい場合は、Object First in entry order 定数を渡します (OBJECT Get pointerコマンド参照)。
- ラジオボタングループへの関連付け: ラジオボタンをグループ化するために使用します。複製したオブジェクトがラジオボタンのとき、boundToに新しいオブジェクトを関連付けたいグループのラジオボタンの名前を渡します。

この引数を省略するか空の文字列を渡すと、新しいオブジェクトはフォームページ中の最後の入力可能オブジェクトとなります。ラジオボタンの場合、オブジェクトはソースボタンのグループに含められます。

新しいオブジェクトはmoveH、moveV、resizeH そしてresizeV 引数を使用して移動およびリサイズできます。OBJECT MOVE コマンドのように、移動やリサイズの方向はmoveH やmoveV 引数に渡された値の符号で指定されます:

- 値が正数の場合、移動やリサイズはそれぞれ右および下方向に行われます。
- 値が負数の場合、移動やリサイズはそれぞれ左および上方向に行われます。

デフォルトでmoveH、moveV、resizeH そしてresizeV の値は、以前の場所からの相対位置で、オブジェクトの座標を変更します。この引数で絶対座標を指定したい場合、最後のオプションの * 引数を渡します。これらの引数を省略すると、新しいオブジェクトはソースオブジェクトの上に重ねて配置されます。

このコマンドはフォームを表示するコンテキストで使用されなければなりません。コマンドは通常On Load フォームイベントやユーザアクション (On Clicked イベント) で実行されます。

Note: On Load フォームイベントがソースオブジェクトに割り当てられているとき、コマンド実行時に複製されたオブジェクトでも生成されます。これにより例えば値の初期化などが行えます。

技術的および論理的な理由により、**OBJECT DUPLICATE**は特定のイベント内では呼び出すことができません。特に:

- オブジェクトメソッド内で生成されるOn Load イベント
- On Unload イベント
- 印刷のコンテキストに関連するイベント (On Header, On Printing Detail等)。オブジェクトを複数回印刷するにはPrint objectコマンドを使用します。

サポートされていないコンテキストでコマンドが呼び出されると、オブジェクトは複製されずに、OK変数に0が設定されます。コマンドが印刷のコンテキストで呼び出されるとエラー-10601が生成されます。

コマンドが正しく実行されるとOKシステム変数に1が、そうでなければ0が設定されます。

例題 1

既存の"OKButton" オブジェクトの上に新しいボタン"CancelButton" を作成し、vCancel 変数を割り当てます:

```
OBJECT DUPLICATE(*;"OKButton";"CancelButton";vCancel)
```

例題 2

既存のラジオボタン"bRadio5" を基に新しいラジオボタン"bRadio6" を作成します。このボタンには変数<>r6 が割り当てられ、"bRadio5" ボタンのと同じグループに入ります。位置は20 ピクセル下に作成されます。:

```
OBJECT DUPLICATE(*;"bRadio5";"bRadio6";<>r6;"bRadio5";0;20)
```

OBJECT Get action

OBJECT Get action ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字列)、省略時: objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (*指定時)、または変数やフィールド (*省略時)
戻り値	テキスト	↩	設定された標準アクション

説明

OBJECT Get actionコマンドは、object引数で指定したオブジェクトに設定された標準アクションのコードを返します。

オブジェクトに対する標準アクションは、デザインモードにおいてプロパティリストで設定するか(*Design Reference* マニュアルの 標準アクション を参照して下さい)、OBJECT SET ACTION コマンドという新しいコマンドを使用することによって設定できます。

このコマンドはオブジェクトに関連付けられた標準アクションのコードを格納した文字列を返します。返された値と対応する定数は"関連付けられた標準アクションのテキスト値" テーマ内にあります。以下の通りです:

定数	型	値
Object Accept action	文字列	2
Object Add subrecord action	文字列	14
Object Automatic splitter action	文字列	16
Object Cancel action	文字列	1
Object Clear action	文字列	21
Object Copy action	文字列	19
Object Cut action	文字列	18
Object Database Settings action	文字列	32
Object Delete record action	文字列	7
Object Delete subrecord action	文字列	13
Object Edit subrecord action	文字列	12
Object First page action	文字列	10
Object First record action	文字列	5
Object Goto page action	文字列	15
Object Last page action	文字列	11
Object Last record action	文字列	6
Object MSC action	文字列	36
Object Next page action	文字列	8
Object Next record action	文字列	3
Object No standard action	文字列	0
Object Open back URL action	文字列	37
Object Open next URL action	文字列	38
Object Paste action	文字列	20
Object Previous page action	文字列	9
Object Previous record action	文字列	4
Object Quit action	文字列	27
Object Redo action	文字列	31
Object Refresh current URL action	文字列	39
Object Return to Design mode action	文字列	35
Object Select all action	文字列	22
Object Show Clipboard action	文字列	23
Object Stop loading URL action	文字列	40
Object Test Application action	文字列	26
Object Undo action	文字列	17

例題

フォーム内にあるオブジェクトのうち、まだ何もアクションが関連付けされてないものに関して全て「キャンセル」のアクションを関連付けたい場合:

```
ARRAY TEXT($arrObjects;0)

FORM GET OBJECTS($arrObjects)
For($i;1;Size of array($arrObjects))
   If(OBJECT Get action(*;$arrObjects{$i})=Object No standard action)
      OBJECT SET ACTION(*;$arrObjects{$i};Object Cancel action)
   End if
End for
```

OBJECT Get auto spellcheck

OBJECT Get auto spellcheck ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	⇒	指定時objectはオブジェクト名 (文字列) 省略時objectは変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) 変数またはフィールド (* 省略時)
戻り値	ブール	↩	True = 自動スペルチェック False = 自動スペルチェックなし

説明

OBJECT Get auto spellcheckコマンドはobjectと* 引数で指定したオブジェクトのカレントプロセスの自動スペルチェックオプションに関する設定値を返します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

このコマンドは自動スペルチェックがobjectに対し有効になっていれば**True**、そうでなければ**False**を返します。

OBJECT GET BEST SIZE

OBJECT GET BEST SIZE ({* ;} object ; bestWidth ; bestHeight {; maxWidth})

引数	型		説明
*	演算子	⇒	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
bestWidth	倍長整数	⇐	最適オブジェクト幅
bestHeight	倍長整数	⇐	最適オブジェクト高さ
maxWidth	倍長整数	⇒	最大オブジェクト幅

説明

OBJECT GET BEST SIZE コマンドは、引数 * と objectで指定されたフォームオブジェクトの"最適な"幅と高さを、引数bestWidth と bestHeightに返します。これらの値はピクセルで表わされます。このコマンドは複雑なレポートの表示や印刷に役立ち、OBJECT MOVEコマンドとともに使用します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。

返される最適値は、カレントのコンテンツが境界内に完全に収まるための、オブジェクトサイズの最小値です。通常これらの値はテキストを含むオブジェクトに 関してのみ意味を持ちます。この計算にはフォント、フォントサイズ、フォントスタイルおよびオブジェクトのコンテンツが考慮されます。さらに、ハイフンや 改行も考慮されます。3Dボタンの場合、コマンドはボタンにアイコンしか表示されない場合でも動作します。
指定されたオブジェクトが空の場合、bestWidthには0が返されます。

返されるサイズは、オブジェクトの周囲に貼り付けられたグラフィックフレームやスクロールバーを計算に入れていません。画面上のオブジェクトの実際のサイズを取得するには、これらの要素の幅を加算する必要があります。

任意のmaxWidth引数により、オブジェクトに最大幅を割り当てることができます。オブジェクトの最適な幅がこの値よりも大きい場合、OBJECT GET BEST SIZEコマンドはbestWidthにmaxWidthを返し、この結果として最適な高さを大きくします。

このコマンドは、次のオブジェクトを処理することができます。

- スタティックテキストエリア
- 参照として挿入されたテキスト
- 次のタイプのフィールドや変数: 文字、テキスト、実数、整数、倍長整数、日付、時間、ブール（チェックボックスとラジオボタン）
- ボタン

この他のオブジェクトタイプ（グループエリア、タブ、矩形、直線、円/楕円、プラグインエリア等）の場合、OBJECT GET BEST SIZEコマンドは現在のオブジェクトサイズ（フォームエディタやOBJECT MOVEコマンドで指定）を返します。

例題

SET PRINT MARKER コマンド.の例を参照。

OBJECT Get border style

OBJECT Get border style ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字列)、省略時: objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (*指定時)、または変数やフィールド (*省略時)
戻り値	倍長整数	↩	境界線スタイル

説明

OBJECT Get border styleコマンドは、object引数で指定したオブジェクトに設定された境界線スタイルを返します。

オブジェクトの境界線スタイルはデザインモードのプロパティリストを使用するか、新コマンド OBJECT SET BORDER STYLE を使用することで設定できます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

コマンドは、境界線スタイルに対応する値を返します。返される値は "Form Objects (Properties)" テーマの中にあります。以下の通りです:

定数	型	値	コメント
Border Dotted	倍長整数	2	オブジェクトの境界線は1ptの点線になります。
Border Double	倍長整数	5	オブジェクトの境界線は二重線(1ピクセル離れた2本の1ptの線)になります。
Border None	倍長整数	0	オブジェクトは境界線を持ちません。
Border Plain	倍長整数	1	オブジェクトの境界線は連続した一本の1ptの線になります。
Border Raised	倍長整数	3	オブジェクトの境界線は浮き上がったような3Dエフェクトになります。
Border Sunken	倍長整数	4	オブジェクトの境界線は沈み込んだような3Dエフェクトになります。
Border System	倍長整数	6	オブジェクトの境界線はシステムのグラフィック仕様に沿ったものになります。

OBJECT Get context menu

OBJECT Get context menu ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
戻り値	ブール	↩	True = コンテキストメニュー有効、False = コンテキストメニュー無効

説明

OBJECT Get context menuコマンドは、引数 object と * で指定したオブジェクトの、"コンテキストメニュー"オプションのカレントの状態を返します。

コンテキストメニューのオプションは、デザインモードのプロパティリストを使用するか、OBJECT SET CONTEXT MENU という新しいコマンドを使用して設定することができます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

object で指定したオブジェクトにおいてコンテキストメニューが使用可能になっている場合は **True** を、そうでない場合には **False** を返します。

OBJECT GET COORDINATES

OBJECT GET COORDINATES ({* ;} object ; left ; top ; right ; bottom)

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
left	倍長整数	←	オブジェクトの左座標
top	倍長整数	←	オブジェクトの上座標
right	倍長整数	←	オブジェクトの右座標
bottom	倍長整数	←	オブジェクトの下座標

説明

OBJECT GET COORDINATESコマンドは、引数 * およびobjectによって指定された、現在のフォームのオブジェクトのleft, top, right および bottomの座標 (ポイント) を返します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。

object にオブジェクト名を渡し、そこでワイルドカード文字 ("@") を使用して1つ以上のオブジェクトを指定する場合、返される座標は関連するすべてのオブジェクトで構成される四角の座標となります。

Note: バージョン6.5からは、文字列に含まれるワイルドカード文字 (@) の取り扱い方を設定することができます。このオプションは、"オブジェクトプロパティ"コマンドに影響を与えます。4D Design Referenceマニュアルを参照してください。

オブジェクトが存在しない場合やコマンドがフォーム内で呼び出されていない場合、座標(0;0;0;0)が返されます。

例題

"button"で始まるすべてのオブジェクトによって形成される長方形の座標を得たい場合を仮定します:

```
OBJECT GET COORDINATES(*;"button@";LEFT;top;RIGHT;bottom)
```

OBJECT Get data source

OBJECT Get data source ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	➡	指定時: object はオブジェクト名(文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	➡	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
戻り値	ポインター	⬅	オブジェクトのカレントデータソースに対するポインター

説明

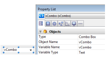
OBJECT Get data sourceコマンドは、引数 object と * で指定したオブジェクトの、カレントのデータソースを返します。

オブジェクトのデータソースはデザインモードのプロパティリストを使用するか、[OBJECT SET DATA SOURCE](#) コマンドを使用することで定義できます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

例題

フォーム内に以下の様に定義されたコンボボックスがあるとします:



Type	Combo Box
Object Name	vCombo
Variable Name	vCombo
Variable Type	Text

以下のコードを実行すると:

```
$vPtr :=OBJECT Get data source(*;"vCombo")
 // $vPtr の値は -> vCombo
```

OBJECT GET DRAG AND DROP OPTIONS

OBJECT GET DRAG AND DROP OPTIONS ({* ;} object ; draggable ; automaticDrag ; droppable ; automaticDrop)

引数	型		説明
*	演算子	⇒	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または変数 (* 省略時)
draggable	ブール	⇐	ドラッグ可能ならTrue、そうでなければFalse
automaticDrag	ブール	⇐	自動ドラッグならTrue、そうでなければFalse
droppable	ブール	⇐	ドロップ可能ならTrue、そうでなければFalse
automaticDrop	ブール	⇐	自動ドロップ可能ならTrue、そうでなければFalse

説明

OBJECT GET DRAG AND DROP OPTIONSコマンドはobjectと* 引数で指定したオブジェクトのカレントプロセスのドラッグ＆ドロップオプションを返します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

このコマンドはデザインモードやOBJECT SET DRAG AND DROP OPTIONSコマンドを使用してカレントプロセスに設定された、現在のドラッグ＆ドロップオプションを返します。

コマンドはデザインモードやOBJECT SET DRAG AND DROP OPTIONSコマンドを使用してカレントプロセスに設定された現在のドラッグ＆ドロップオプションを返します。

各パラメーターは対応するオプションが有効か無効かによってTrueまたはFalseを返します:

- draggable = True: プログラムモードでオブジェクトのドラッグが可能
- automaticDrag = True (テキストフィールドや変数、コンボボックス、リストボックスでのみ使用): 自動モードでオブジェクトのドラッグが可能
- droppable = True: プログラムモードでオブジェクトのドロップが可能
- automaticDrop = True (テキストフィールドや変数、コンボボックス、リストボックスでのみ使用): 自動モードでオブジェクトのドロップが可能

OBJECT Get enabled

OBJECT Get enabled ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字) 省略時: objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
戻り値	ブール	↩	True = オブジェクトは有効; そうでなければFalse

説明

OBJECT Get enabled コマンドは objectで指定されたオブジェクトまたはオブジェクトグループがフォーム中で有効ならTrueを、無効ならFalseを返します。

有効なオブジェクトはマウスクリックやキーボードショートカットに反応します。

オプションの * 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectは変数です。この場合、文字ではなく変数への参照 (変数オブジェクトのみ) を渡します。

このコマンドは以下のタイプのオブジェクトに適用できます:

- ボタン、デフォルトボタン、3Dボタン、非表示ボタン、ハイライトボタン
- ラジオボタン、3Dラジオボタン、ピクチャボタン
- チェックボックス、3Dチェックボックス
- ポップアップメニュー、ドロップダウンリスト、コンボボックス、メニュー/ドロップダウンリスト
- サーモメータ、ルーラ

OBJECT Get enterable

OBJECT Get enterable ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	➡	指定時: objectはオブジェクト名 (文字) 省略時: objectは変数またはフィールド
object	フォームオブジェクト	➡	オブジェクト名 (* 指定時) または フィールドまたは変数 (* 省略時)
戻り値	ブール	⮐	True = 入力可; そうでなければ false

説明

OBJECT Get enterable コマンドはobjectで指定されたオブジェクトまたはオブジェクトグループが**入力可属性**を持つ場合にTrueを、そうでなければFalseを返します。

オプションの * 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectはフィールドまたは変数です。この場合、文字ではなくフィールドまたは変数への参照 (フィールドまたは変数オブジェクトのみ) を渡します。

OBJECT GET EVENTS

OBJECT GET EVENTS ({* ;} object ; arrEvents)

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
arrEvents	倍長整数配列	←	Array of enabled events

説明

OBJECT GET EVENTSコマンドは、引数 objectと * で指定したオブジェクトの、フォームイベントのカレントの設定を取得します。

フォームイベントはプロパティリストを通して有効化/無効化することができます。また、同一のプロセスにおいてなら OBJECT SET EVENTS コマンドを使用して設定することも可能です。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。
フォーム自体のイベントの設定を取得したい場合は、任意の * 演算子を渡したうえで、object 引数に空の文字列 "" を渡します。こうすることでカレントフォームをこれで指定します。

注: テーブルに関連付けられたサブフォームのイベントを取得したい場合、オブジェクト名を使用する記法しか使用できません。

arrEvents 引数には、倍長整数配列を渡します。この配列はコマンドが実行されたときに自動的にリサイズされ、オブジェクトまたはフォームの中で有効化されている、定義済みのイベントかカスタムイベントのコードを全て受け取ります。この値と対応する定数は "Form Events" テーマ内にあります。

オブジェクトまたはフォームに対してメソッドが何も関連付けられていない場合は、arrEvents は空で返される点に注意して下さい。

例題

イベントを2つ有効化して、オブジェクトのイベントのリストを取得したい場合:

```
ARRAY LONGINT($ArrCurrentEvents;0)
ARRAY LONGINT($ArrEnabled;2)
$ArrEnabled{1}:=On Header Click
$ArrEnabled{2}:=On Footer Click
OBJECT SET EVENTS(*;"Col1";$ArrEnabled;Enable events others unchanged)
OBJECT GET EVENTS(*;"Col1";$ArrCurrentEvents)
```

OBJECT Get filter

OBJECT Get filter ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字) 省略時: objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または フィールドまたは変数 (* 省略時)
戻り値	テキスト	↩	フィルター名

説明

OBJECT Get filter コマンドは objectで指定されたオブジェクトまたはオブジェクトグループに割り当てられたフィルターの名前を返します。.

オプションの * 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectはフィールドまたは変数です。この場合、文字ではなくフィールドまたは変数への参照 (フィールドまたは変数オブジェクトのみ) を渡します。

OBJECT Get focus rectangle invisible

OBJECT Get focus rectangle invisible ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) 変数またはフィールド (* 省略時)
戻り値	倍長整数	←	True = フォーカスの四角を隠す False = フォーカスの四角を表示する

説明

OBJECT Get focus rectangle invisibleコマンドはobjectと * 引数で指定したオブジェクトのフォーカスの四角に関するカレントプロセス内の表示オプションを返します。この設定は、フォームエディターのプロパティリストの入力可能オブジェクトで利用できる**フォーカスの四角を隠す**オプションに対応します。このコマンドはデザインモードやOBJECT SET FOCUS RECTANGLE INVISIBLEコマンドを使用して指定されたこのオプションに関する現在の設定値を返します。

注: このオプションはMac OSでのみ利用できます。Windowsでは効果がありません。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

このコマンドはフォーカスの四角が隠されていると**True**を、表示されていると**False**を返します。

OBJECT Get font

OBJECT Get font ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字)、省略時: objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または、フィールドまたは変数 (* 省略時)
戻り値	テキスト	↩	フォント名

説明

OBJECT Get fontコマンドは objectで指定されたフォームオブジェクトで使用されている文字フォントの名前を返します。

オプションの* 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectはフィールドまたは変数です。この場合、文字ではなくフィールドまたは変数への参照 (フィールドまたは変数オブジェクトのみ) を渡します。

OBJECT Get font size

OBJECT Get font size ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	➡	指定時: objectはオブジェクト名 (文字)、省略時: objectは変数またはフィールド
object	フォームオブジェクト	➡	オブジェクト名 (* 指定時) または、フィールドまたは変数 (* 省略時)
戻り値	倍長整数	⬅	ポイント単位のフォントサイズ

説明

OBJECT Get font size コマンドはobjectで指定されたフォームオブジェクトで使用されている文字フォントのサイズをポイント単位で返します。.

オプションの* 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectはフィールドまたは変数です。この場合、文字ではなくフィールドまたは変数への参照 (フィールドまたは変数オブジェクトのみ) を渡します。

OBJECT Get font style

OBJECT Get font style (* ; object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字) 、省略時: objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時)、またはフィールドまたは変数 (* 省略時)
戻り値	倍長整数	↩	フォントスタイル

説明

OBJECT Get font style コマンドは、objectで指定されたフォームオブジェクトで使用されている文字フォントの現在のスタイルを返します。.

オプションの * 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectはフィールドまたは変数です。この場合、文字ではなくフィールドまたは変数への参照 (フィールドまたは変数オブジェクトのみ) を渡します。

返される値を"Font Styles"テーマの定義済み定数と比較することができます:

定数	型	値
Plain	倍長整数	0
Bold	倍長整数	1
Italic	倍長整数	2
Underline	倍長整数	4

OBJECT Get format

OBJECT Get format ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
戻り値	文字	↩	オブジェクト表示フォーマット

説明

OBJECT Get format コマンドは、object引数で指定されたオブジェクトに適用された現在の表示フォーマットを返します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。

このコマンドはオブジェクトの現在の表示フォーマット、つまりデザインモードやOBJECT SET FORMATコマンドで定義されたフォーマットを返します。OBJECT Get formatは、表示フォーマットを受け入れるあらゆるタイプのフォームオブジェクト (フィールドや変数) に対して使用することができます (ブール、日付、時間、ピクチャ、文字列、数値、ボタングリッド、ダイヤル、サーモメータ、ルーラ、ピクチャポップアップメニュー、ピクチャボタン、3Dボタン)。これらのオブジェクトの表示フォーマットに関する詳細は、OBJECT SET FORMATコマンドの説明を参照してください。

Note: コマンドを一連のオブジェクトに対して適用した場合、最後に選択されたオブジェクトのフォーマットが返されます。

OBJECT Get format コマンドを日付、時間、ピクチャタイプのオブジェクト (定数で定義されたフォーマット) に対して適用すると、定数の文字コードに相当する文字列が返されます。定数の値を取得するには、この戻り値に対してCharacter codeコマンドを適用してください（後述）。

例題 1

この例は、"myphoto"という名前のピクチャ変数に対して適用されたフォーマット定数の値を取得します:

```
C_STRING(2;$format)
OBJECT SET FORMAT(*;"myphoto";Char(On Background))
 //背景フォーマットを適用する (value = 3)
$format:=OBJECT Get format(*;"myphoto")
ALERT("フォーマット番号:"+String(Character code($format)))
 //"3"が表示される
```

例題 2

この例は、ブールフィールド"[Members]Marital_status"に対して適用されたフォーマットを取得することができます:

```
C_STRING(30;$format)
$format:=OBJECT Get format([Members]Marital_status)
ALERT($format) //表示フォーマット, 例えば"Married;Single"
```

OBJECT Get help tip

OBJECT Get help tip ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	⇒	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または変数 (* 省略時)
戻り値	テキスト	⮐	オブジェクトのヘルプメッセージ

説明

OBJECT Get help tipコマンドはobject と* 引数で指定したオブジェクトに割り当てられたカレントプロセスのヘルプメッセージを返します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

このコマンドは指定したオブジェクトにデザインモードあるいはOBJECT SET HELP TIPコマンドでカレントプロセスに関連付けられた、現在のヘルプメッセージを返します。返される文字列はフォームが実行されたときに表示されるメッセージです。4D参照やxliff resnameなどの可変な値が含まれる場合、文脈に基づき解析された結果が返されます。

例題

ピクチャーボタンのタイトルがヘルプメッセージとして格納されています。このタイトルはxliffファイルに記述されていて、アプリケーションのカレント言語に基づき変わります:

```
OBJECT SET HELP TIP(*;"button1";":xliff:btn_yes")
$helpmessage:=OBJECT Get help tip(*;"button1")
 // $helpmessageには例えば日本語環境では"はい"、英語環境では"YES"が返されます (xliffにそのように記述されていれば)。
```

OBJECT Get horizontal alignment

OBJECT Get horizontal alignment ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
戻り値	倍長整数	↩	整列コード

説明

OBJECT Get horizontal alignmentコマンドは、引数objectおよび * で指定されたオブジェクトに適用された整列タイプを示すコードを返します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。

注: オブジェクトグループにこのコマンドを適用すると、最後のオブジェクトの整列コードのみが返されます。

返されるコードはForm Objects (Properties)テーマ内にある以下の定数のいずれかとなります:

定数	型	値
Align center	倍長整数	3
Align default	倍長整数	1
Align left	倍長整数	2
Align right	倍長整数	4

整列を適用できるフォームオブジェクトは次の通りです:

- コンボボックス
- スタティックテキスト
- グループエリア
- ポップアップメニュー/ドロップダウンリスト
- フィールド
- 変数
- リストボックス
- リストボックス列
- リストボックスヘッダー
- リストボックスフッター

OBJECT Get indicator type

OBJECT Get indicator type ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
戻り値	倍長整数	↩	インジケータータイプ

説明

OBJECT Get indicator typeコマンドは、引数 object と * で指定したサーモメーターに割り当てられた、カレントのインジケーターの型を返します。

インジケーターのタイプは、デザインモードのプロパティリストを使用するか、新コマンド OBJECT SET INDICATOR TYPE を使用することによって定義できます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

返された値は、"Form Objects (Properties)" テーマ内にある以下の定数と対応しています:

定数	型	値	コメント
Asynchronous progress bar	倍長整数	3	連続したアニメーションを表示する、回転型のインジケーター
Barber shop	倍長整数	2	連続したアニメーションを表示するバー
Progress bar	倍長整数	1	標準の進捗バー

OBJECT Get keyboard layout

OBJECT Get keyboard layout ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) 変数またはフィールド (* 省略時)
戻り値	文字	↩	レイアウトのランゲージコード、"" = レイアウトなし

説明

OBJECT Get keyboard layoutコマンドはobjectと * 引数で指定したオブジェクトにカレントプロセスで割り当てられたカレントキーボードレイアウトを返します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

このコマンドは使用する言語の (RFC3066、ISO639、そしてISO3166に基づく) コードを示す文字列を返します。詳細はSET DATABASE LOCALIZATIONコマンドの説明を参照してください。

OBJECT Get list name

OBJECT Get list name ({* ;} object {; listType}) -> 戻り値

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字) 省略時: objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または フィールドまたは変数 (* 省略時)
listType	倍長整数	→	Type of list: Choice list, Required list or Excluded list
戻り値	テキスト	↩	(デザインモードで指定された) 選択リストの名前

説明

OBJECT Get list name コマンドは objectで指定されたオブジェクトまたはオブジェクトグループに割り当てられた選択リストの名前を返します。4Dではフォームエディタあるいは OBJECT SET LIST BY NAME コマンドを使用してフォームオブジェクトに (デザインモードのリストエディタで作成された) 選択リストを割り当てることができます。

オプションの * 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectはフィールドまたは変数です。この場合、文字ではなくフィールドまたは変数への参照 (フィールドまたは変数オブジェクトのみ) を渡します。

任意の listType 引数を使用して取得したいリストの型を指定することもできます。この引数を省略した場合、コマンドはデフォルトでオブジェクトに関連付けられた選択リスト(値のリスト)の名前を返します。"Form Objects (Properties)"テーマ内にある以下のいずれか一つの定数を listType 引数に渡すことによって、指定リストや除外リストの名前を取得することもできます:

定数	型	値	コメント
Choice list	倍長整数	0	選択できる値のリスト(プロパティリスト内の「選択リスト」)。(デフォルト)
Excluded list	倍長整数	2	入力できない値のリスト。(プロパティリスト内の「除外リスト」)
Required list	倍長整数	1	入力可能な値のリスト(プロパティリストの"指定リスト"オプション)。

object で指定されたオブジェクトに指定されたリストが関連付けられていない場合、コマンドは空の文字列("")を返します。

OBJECT Get list reference

OBJECT Get list reference ({* ;} object {; listType}) -> 戻り値

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
listType	倍長整数	→	リストの種類:選択リスト、指定リスト、除外リスト
戻り値	ListRef	←	リストの参照番号

説明

OBJECT Get list referenceコマンドは、引数 objectと * で指定したオブジェクトまたはオブジェクトのグループに関連付けられた階層リストの参照番号(ListRef)を返します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

listType 引数を省略した場合、コマンドは自動的にオブジェクトに割り当てられた選択リストの参照番号を返します。また、listType 引数に "Form Objects (Properties)" テーマ内にある以下の定数を渡すことにより、指定リストと除外リストの参照番号を取得することが出来ます:

定数	型	値	コメント
Choice list	倍長整数	0	選択できる値のリスト(プロパティリスト内の「選択リスト」)。(デフォルト)
Excluded list	倍長整数	2	入力できない値のリスト。(プロパティリスト内の「除外リスト」)
Required list	倍長整数	1	入力可能な値のリスト(プロパティリストの"指定リスト"オプション)。

listType で指定した種類の階層リストが関連付けされていない場合、コマンドは 0 を返します。

OBJECT GET MAXIMUM VALUE

OBJECT GET MAXIMUM VALUE ({* ;} object ; maxValue)

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
maxValue	日付, 時間, Number	⇐	オブジェクトのカレントの最大値

説明

OBJECT GET MAXIMUM VALUEコマンドは、引数 object と * で指定したオブジェクトのカレントの最大値を、maxValue 変数の中に返します。

最大値のプロパティは、デザインモードのプロパティリストを使用するか、新コマンド OBJECT SET MAXIMUM VALUE を使用することによって定義できます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

OBJECT GET MINIMUM VALUE

OBJECT GET MINIMUM VALUE ({* ;} object ; minValue)

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
minValue	日付, 時間, Number	⇐	オブジェクトのカレントの最小値

説明

OBJECT GET MINIMUM VALUEコマンドは、引数 objectと * で指定したオブジェクトの最小値を、minValue 変数の中に返します。

最小値のプロパティは、デザインモードのプロパティリストを使用するか、新コマンド OBJECT SET MINIMUM VALUE を使用することによって定義できます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

OBJECT Get multiline

OBJECT Get multiline ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
戻り値	倍長整数	←	複数行の状態

説明

OBJECT Get multilineコマンドは、引数 object と * で指定したオブジェクトの、"複数行"のオプションのカレントの状態を返します。

「複数行」のオプションはデザインモードのプロパティリストを使用するか、OBJECT SET MULTILINE という新しいコマンドを使用して設定することができます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

返ってきた値は "Form Objects (Properties)" テーマ内にある以下の定数と対応しています:

定数	型	値	コメント
Multiline Auto	倍長整数	0	単独行のエリアでは、行に表示しきれない単語は切り落とされ、改行はされません。 複数行のエリアでは、改行が行われます。
Multiline No	倍長整数	2	改行は禁止されます。テキストは必ず単独行として表示されます。文字列かテキストフィールドか変数に改行が含まれていたとしても、改行は行われません。
Multiline Yes	倍長整数	1	単独行のエリアでは、テキストは最初の改行までか、単語全体を表示できる最後の単語までが表示されます。その後改行が挿入されるので、↓キーを押すことによってエリアの内容をスクロールすることができます。 複数行のエリアでは、自動で改行が行われます。

注: OBJECT Get multiline コマンドを、複数行のオプションがないオブジェクトに対して使用した場合、コマンドは 0 を返します。

OBJECT Get placeholder

OBJECT Get placeholder ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
戻り値	テキスト	↩	オブジェクトと関連付けられたプレースホルダーテキスト

説明

OBJECT Get placeholderコマンドは、引数 object と * で指定したオブジェクトと関連付けられたプレースホルダーテキストを返します。オブジェクトにプレースホルダーテキストが何も関連付けられていない場合は、空の文字列を返します。

プレースホルダーテキストは、プロパティリストを使用するか、OBJECT SET PLACEHOLDER コマンドを使用することで定義できます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

プレースホルダーにプロパティリストを通じてxliff参照が定義されていた場合、コマンドは参照値ではなく、":xliff:*resname*"という原文をそのまま返します。

例題

あるフィールドのプレースホルダーのテキストを取得したい場合:

```
$txt:=OBJECT Get placeholder([People]LastName)
```

OBJECT GET PRINT VARIABLE FRAME

OBJECT GET PRINT VARIABLE FRAME ({* ;} object ; variableFrame {; fixedSubform})

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
variableFrame	ブール	⇐	True = 可変長フレームを使用、False = 固定長フレームを使用
fixedSubform	倍長整数	⇐	固定長フレーム時のサブフォームに対するオプション

説明

OBJECT GET PRINT VARIABLE FRAMEコマンドは、引数 objectと * で指定したオブジェクトの印刷時可変オプションのカレントの設定を取得します。

フレーム印刷のプロパティはプロパティリストか、OBJECT SET PRINT VARIABLE FRAME という新しいコマンドを使用することで設定できます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

variableFrame 引数には、フレーム印刷のオプションの状態がブールで返されます。**True** の場合は「可変」を、**False** の場合は「固定」を意味します。

object で指定したオブジェクトがサブフォームで、フレーム印刷が「固定」に設定されている場合に限り、コマンドは fixedSubform 引数に固定フレーム印刷時のオプションの状態を返します。返される値は "Form Objects (Properties)" テーマ内にあります。以下の通りです:

定数	型	値	コメント
Print Frame fixed with multiple records	倍長整数	2	フレームは同じサイズを維持しますが、4Dは全てのレコードが載るまで複数回フォームを印刷します。
Print Frame fixed with truncation	倍長整数	1	4Dはサブフォームのエリアに収まるレコードのみ印刷します。フォームは一度だけ印刷され、印刷されなかったレコードは無視されます。

OBJECT GET RESIZING OPTIONS

OBJECT GET RESIZING OPTIONS ({* ;} object ; horizontal ; vertical)

引数	型		説明
*	演算子	⇒	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または変数 (* 省略時)
horizontal	倍長整数	⇐	横リサイズオプション
vertical	倍長整数	⇐	縦リサイズオプション

説明

OBJECT GET RESIZING OPTIONSコマンドはobjectと* 引数で指定したオブジェクトのカレントプロセスのリサイズオプションを返します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

このコマンドはデザインモードやOBJECT SET RESIZING OPTIONSコマンドを使用してプロセスに設定された、リサイズに関するオプションの現在の設定値を返します。これらのオプションはフォームウィンドウのサイズが変更されたときのオブジェクトの表示方法を規定します。

horizontal 引数には横方向のリサイズオプション設定値が返されます。この値をForm Objects (Properties)テーマの以下の定数と比較できます:

定数	型	値	コメント
Resize horizontal grow	倍長整数	1	ウィンドウが横方向に拡げられたら、オブジェクトの幅も同じ比率だけ右に拡大する。
Resize horizontal move	倍長整数	2	ウィンドウの幅が拡げられたら、オブジェクトも同じだけ右方向に移動する。
Resize horizontal none	倍長整数	0	ウィンドウの幅が変更されても、オブジェクトの位置及びサイズを変更しない。

vertical 引数には縦方向のリサイズオプション設定値が返されます。この値をForm Objects (Properties)テーマの以下の定数と比較できます:

定数	型	値	コメント
Resize vertical grow	倍長整数	1	ウィンドウが縦方向に拡げられたら、オブジェクトの高さも同じ比率だけ下に拡大する。
Resize vertical move	倍長整数	2	ウィンドウの高さが拡げられたら、オブジェクトも同じだけ下方向に移動する。
Resize vertical none	倍長整数	0	ウィンドウの高さが変更されても、オブジェクトの位置及びサイズを変更しない。

OBJECT GET RGB COLORS

OBJECT GET RGB COLORS (* ; object ; foregroundColor ; backgroundColor ; altBackgrndColor)

引数	型	説明
*	演算子	→ 指定時: objectはオブジェクト名 (文字) 省略時: objectは変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名 (* 指定時) または フィールドまたは変数 (* 省略時)
foregroundColor	倍長整数	← 描画色のRGBカラー値
backgroundColor	倍長整数	← 背景色のRGBカラー値
altBackgrndColor	倍長整数	← 奇数行の色のRGBカラー値

説明

OBJECT GET RGB COLORS コマンドはobjectで指定されたオブジェクトまたはオブジェクトグループの描画色や背景色を返します。.

オプションの* 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectはフィールドまたは変数です。この場合、文字ではなくフィールドまたは変数への参照 (フィールドまたは変数オブジェクトのみ) を渡します。

このコマンドをリストボックスタイプのオブジェクトに適用すると、altBackgrndColor引数に奇数行のカラー値が返されることがあります。この場合、backgroundColorの値は偶数行にのみ使用されます。

foregroundColor、backgroundColor、そしてaltBackgrndColor引数に返されるRGBカラー値は4バイトの倍長整数値で0x00RRGGBBのフォーマット、またはシステムカラーに対応する負数です。後者の場合、取得した値をSET RGB COLORSテーマの定数と比較できます:

定数	型	値	コメント
Background color	倍長整数	-2	
Background color none	倍長整数	-16	この定数は *backgroundColor* 引数と *altBackgrndColor* 引数にしか渡すことができません。
Dark shadow color	倍長整数	-3	
Disable highlight item color	倍長整数	-11	
Foreground color	倍長整数	-1	
Highlight menu background color	倍長整数	-9	
Highlight menu text color	倍長整数	-10	
Highlight text background color	倍長整数	-7	
Highlight text color	倍長整数	-8	
Light shadow color	倍長整数	-4	

注: システムカラーはOBJECT SET RGB COLORSコマンドを使用して適用されます。

foregroundColor、backgroundColor、およびaltBackgrndColor引数のフォーマットに関する詳細はOBJECT SET RGB COLORSコマンドを参照してください。

OBJECT GET SCROLL POSITION

OBJECT GET SCROLL POSITION ({* ;} object ; vPosition {; hPosition})

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字)、省略時: objectは変数、フィールドまたはテーブル
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数、フィールドまたはテーブル (* 省略時)
vPosition	倍長整数	←	表示されている最初の行数、または ピクセル単位の縦スクロール (ピクチャ)
hPosition	倍長整数	←	表示されている最初の列数、または ピクセル単位の横スクロール (ピクチャ)

説明

OBJECT GET SCROLL POSITION はobjectで指定されたフォームオブジェクトの、スクロールバーの位置に関連する情報をvPositionとhPosition引数に返します。

オプションの* 引数を渡すと、object引数はサブフォーム、階層リスト、スクロールエリア、リストボックス、またはピクチャタイプのオブジェクト名 (文字) です。この引数を渡さないと、objectはテーブル (リストフォームまたはサブフォームテーブル) 、変数 (階層リストのListRef、ピクチャまたはリストボックス変数) またはフィールドです。

objectがリストタイプのオブジェクト (サブフォーム、リストフォーム、階層リスト、スクロールエリア、またはリストボックス) を指定する場合、vPositionにはobject中で表示されている最初の行の番号が返されます。リストボックスの場合のみ、hPositionにはリストボックス中で一番左に完全に表示されている列の番号が返されます。他のタイプのオブジェクトの場合、この引数には0が返されます。

objectがピクチャ (変数またはフィールド) をさす場合、vPositionには縦移動、hPositionには横移動が返されます。これらの値はピクセル単位で表現され、ローカル座標システムのピクチャの基点を0とします。

OBJECT GET SCROLLBAR

OBJECT GET SCROLLBAR ({* ;} object ; horizontal ; vertical)

引数	型		説明
*	演算子	⇒	指定時: objectはオブジェクト名 (文字)、省略時: objectは変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または、フィールドまたは変数 (* 省略時)
horizontal	ブール	⇐	True=表示, False=非表示
vertical	ブール	⇐	True=表示, False=非表示

説明

OBJECT GET SCROLLBAR コマンドは、objectで指定されたオブジェクトまたはオブジェクトグループの縦横スクロールバーの表示/非表示状態を知るために使用します。.

オプションの * 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectはフィールドまたは変数です。この場合、文字ではなくフィールドまたは変数への参照 (フィールドまたは変数オブジェクトのみ) を渡します。

このコマンドは以下のオブジェクトに対して使用できます:

- リストボックス
- スクロールエリア
- 階層リスト
- サブフォーム

OBJECT GET SHORTCUT

OBJECT GET SHORTCUT ({* ;} object ; key ; modifiers)

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
key	文字	←	オブジェクトに割り当てられたキー
modifiers	倍長整数	←	モディファイアーキーマスクまたはマスクの組み合わせ

説明

OBJECT GET SHORTCUTコマンドはobjectと*引数で指定されたオブジェクトに割り当てられたカレントプロセスのキーボードショートカットを返します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

key 引数にはそのオブジェクトに割り当てられたキー (標準キーの場合)、またはキーの値を示すブラケットで囲まれた文字列 (ファンクションキーの場合) が返されます。この値をShortcut and Associated Keysテーマの定数と比較できます (OBJECT SET SHORTCUTコマンド参照)。

modifiers 引数にはそのショートカットに関連付けられたモディファイアーキーを表す値が返されます。複数のモディファイアーキーが組み合わせて指定されている場合、コマンドはその合計値を返します。この値をEvents (Modifiers)テーマの以下の定数と比較できます:

定数	型	値	コメント
Command key mask	倍長整数	256	Windows = Ctrlキー、Mac OS = Commandキー
Control key mask	倍長整数	4096	Mac OSのみ
Option key mask	倍長整数	2048	Windows = Altキー、Mac OS = Optionキー
Shift key mask	倍長整数	512	WindowsおよびMac OS

ショートカットにモディファイアーキーが関連付けられていない場合、modifiers には0が返されます。

注: object 引数がフォーム中の複数のオブジェクトに該当し、それらに異なる値が設定されている場合、key には空の文字列 ("") が、そして modifiers には0が返されます。

OBJECT Get style sheet

OBJECT Get style sheet ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
戻り値	テキスト	↩	スタイルシート名

説明

OBJECT Get style sheetコマンドは、引数 objectと * で指定したオブジェクトに関連付けられたスタイルシートの名前を返します。

スタイルシートはデザインモードにおいてプロパティリストで設定することができます。または、セッションの間においては、OBJECT SET STYLE SHEETコマンドを使用して設定することもできます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

コマンドから以下のどれかが返されます:

- スタイルシート名
- 自動スタイルシートが設定されていた場合は定数 "__automatic__" が返されます。
- スタイルシートが何も設定されていない場合は空の文字列 ("") が返されます。

コマンドで複数のオブジェクトを指定した場合、コマンドが最初に見つけたオブジェクトのスタイルシートのみが返されます。

OBJECT GET SUBFORM

OBJECT GET SUBFORM ({* ;} object ; tablePtr ; detailSubform {; listSubform})

引数	型		説明
*	演算子	⇒	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または変数 (* 省略時)
tablePtr	テーブル	⇐	フォームが属するテーブルへのポインター
detailSubform	テキスト	⇐	サブフォームの詳細フォーム名
listSubform	テキスト	⇐	サブフォームのリストフォーム名 (テーブルフォーム)

説明

OBJECT GET SUBFORMコマンドはobjectと*引数で指定したサブフォームオブジェクトに関連付けられたフォームの名前を返します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

tablePtr 引数には使用されるフォームが属するテーブルへのポインターが返されます。サブフォームにプロジェクトフォームが指定されている場合、この引数にはNilになります。

detailSubform 引数にはサブフォームで使用されている詳細フォーム名が返されます。

listSubform 引数にはサブフォームで使用されているリストフォーム名が返されます。リストフォームが指定されていない場合空の文字列が返されます。

OBJECT GET SUBFORM CONTAINER SIZE

OBJECT GET SUBFORM CONTAINER SIZE (width ; height)

引数	型		説明
width	倍長整数	⇐	サブフォームオブジェクトの幅
height	倍長整数	⇐	サブフォームオブジェクトの高さ

説明

OBJECT GET SUBFORM CONTAINER SIZEコマンドは親フォーム中に表示されているカレントサブフォームオブジェクトの幅と高さをピクセル単位で返します。

このコマンドはサブフォームとして使用され、サブフォームオブジェクトの中に表示されているフォームのメソッドから呼び出されなければなりません。コマンドはサブフォームを含んでいるオブジェクトの幅と高さを返します。

このコマンドは例えば、サブフォームオブジェクトの特性に基づきサブフォームオブジェクトのサイズを変更したり移動したりしなければならないようなケースで使用できます。On Load や On Resize フォームイベントでサブフォームはこのコマンドを呼び出し、自身のオブジェクトを表示するためのスペースを計算できます。

- コマンドがサブフォームとして使用されていないフォームから呼び出された場合、コマンドはフォームウィンドウの現在のサイズを返します。
- 画面表示が関連しない状況でコマンドが呼び出された場合 (例えば印刷時)、width と height には0が返されます。

OBJECT Get text orientation

OBJECT Get text orientation ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
戻り値	倍長整数	⇐	テキストの回転角度

説明

OBJECT Get text orientationコマンドは、引数 object と * で指定したオブジェクトのテキストに適用されたカレントの方向の値を返します。

「方向」のオプションは、デザインモードのプロパティリストを使用するか、OBJECT SET TEXT ORIENTATION という新しいコマンドを使用して設定することができます。

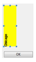
任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

返ってきた値は "Form Objects (Properties)" テーマ内にある以下の定数と対応しています:

定数	型	値	コメント
Orientation 0°	倍長整数	0	回転なし(デフォルト値)
Orientation 180°	倍長整数	180	テキストの方向を時計回りに180°回転
Orientation 90° left	倍長整数	270	テキストの方向を反時計回りに90°回転
Orientation 90° right	倍長整数	90	テキストの方向を時計回りに90°回転

例題

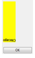
以下の様な、フォームエディターで"90°反時計回り"が適用されているオブジェクトがあるとき:



フォームを実行した際、以下のコマンドを実行すると:

```
OBJECT SET TEXT ORIENTATION(*;"myText";Orientation 180°)
```

... オブジェクトは以下のように表示されます。:



```
$vOrt:=OBJECT Get text orientation(*;"myText") //$vOrt=180
```

OBJECT Get three states checkbox

OBJECT Get three states checkbox ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
戻り値	ブール	↩	True = スリーステートチェックボックス、False = 標準のチェックボックス

説明

OBJECT Get three states checkboxコマンドは、引数 objectと * で指定したチェックボックスの"スリーステート"プロパティの、カレントの状態を返します。

スリーステートのプロパティは、プロパティリストを使用するか、同一プロセスにおいてであれば OBJECT SET THREE STATES CHECKBOX を呼び出して設定することができます。

OBJECT Get title

OBJECT Get title ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	⇒	指定時: objectはオブジェクト名 (文字)、省略時: objectは変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または、フィールドまたは変数 (* 省略時)
戻り値	テキスト	↩	ボタンのタイトル

説明

OBJECT Get titleコマンドは objectで指定されたフォームオブジェクトのタイトル (ラベル) を返します。このコマンドはラベルを表示するすべてのタイプのシンプルオブジェクトに使用できます:

- ボタン
- チェックボックス
- ラジオボタン
- スタティックテキスト
- グループボックス

オプションの* 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectはフィールドまたは変数です。この場合、文字ではなくフィールドまたは変数への参照 (フィールドまたは変数オブジェクトのみ) を渡します。

OBJECT Get type

OBJECT Get type ({* ;} object) -> 戻り値

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
戻り値	倍長整数	↩ オブジェクトのタイプ

説明

OBJECT Get typeコマンドは、カレントフォーム内の、引数 objectと * で指定したオブジェクトのタイプを返します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。線や長方形といった静的なオブジェクトを処理する場合にはこの記法を用いなければなりません。省略時には object 引数でフィールドまたは変数を指定します。

注: このコマンドを複数のオブジェクトに対して適用した場合、最背面にあるオブジェクトのタイプが返されます。

返された値は "Form Object Types" テーマ内にある以下の定数と対応しています:

定数	型	値
Object type 3D button	倍長整数	16
Object type 3D checkbox	倍長整数	26
Object type 3D radio button	倍長整数	23
Object type button grid	倍長整数	20
Object type checkbox	倍長整数	25
Object type combobox	倍長整数	11
Object type dial	倍長整数	28
Object type group	倍長整数	21
Object type groupbox	倍長整数	30
Object type hierarchical list	倍長整数	6
Object type hierarchical popup	倍長整数	13
Object type highlight button	倍長整数	17
Object type invisible button	倍長整数	18
Object type line	倍長整数	32
Object type listbox	倍長整数	7
Object type listbox column	倍長整数	9
Object type listbox footer	倍長整数	10
Object type listbox header	倍長整数	8
Object type matrix	倍長整数	35
Object type oval	倍長整数	34
Object type picture button	倍長整数	19
Object type picture input	倍長整数	4
Object type picture popup	倍長整数	14
Object type picture radio button	倍長整数	24
Object type plugin area	倍長整数	38
Object type popup drop down	倍長整数	12
Object type progress indicator	倍長整数	27
Object type push button	倍長整数	15
Object type radio button	倍長整数	22
Object type radio button field	倍長整数	5
Object type rectangle	倍長整数	31
Object type rounded rectangle	倍長整数	33
Object type ruler	倍長整数	29
Object type splitter	倍長整数	36
Object type static picture	倍長整数	2
Object type static text	倍長整数	1
Object type subform	倍長整数	39
Object type tab control	倍長整数	37
Object type text input	倍長整数	3
Object type unknown	倍長整数	0
Object type web area	倍長整数	40

例題

フォームをロードし、そこに内包される全てのリストボックスオブジェクトの一覧を取得する場合:

```
FORM LOAD("MyForm")
ARRAY TEXT(arrObjects;0)
FORM GET OBJECTS(arrObjects)
ARRAY LONGINT(ar_type;Size of array(arrObjects))
ARRAY TEXT(ar_typeName;Size of array(arrObjects))
For($i;1;Size of array(arrObjects))
   ar_type{$i}:=OBJECT Get type(*;arrObjects{$i})
   If(ar_type{$i}=Object type listbox)
      ARRAY TEXT(arrLBObjects;0)
      LISTBOX GET OBJECTS(*;arrObjects{$i};arrLBObjects)
   End if
End for
FORM UNLOAD
```

OBJECT Get vertical alignment

OBJECT Get vertical alignment ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
戻り値	倍長整数	↩	行揃えのタイプ

説明

OBJECT Get vertical alignmentコマンドはobject と* 引数で指定したオブジェクトのテキスト縦位置タイプを示す値を返します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

注: このコマンドの一度の呼び出しを複数のオブジェクトに対して実行した場合、最後のオブジェクトの縦位置値のみが返されます。

返される値はForm Objects (Properties)テーマの以下のいずれかの定数に対応します:

定数	型	値
Align bottom	倍長整数	4
Align center	倍長整数	3
Align top	倍長整数	2

縦位置は以下のタイプのフォームオブジェクトに設定できます:

- リストボックス
- リストボックス列
- リストボックスヘッダーおよびフッター

OBJECT Get visible

OBJECT Get visible ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字) 省略時: objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または フィールドまたは変数 (* 省略時)
戻り値	ブール	←	True = オブジェクトは表示; そうでなければFalse

説明

OBJECT Get visible コマンドは objectで指定されたオブジェクトまたはオブジェクトグループが 表示属性を持っていればTrueを、そうでなければFalseを返します。

オプションの * 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectはフィールドまたは変数です。この場合、文字ではなくフィールドまたは変数への参照 (フィールドまたは変数オブジェクトのみ) を渡します。

OBJECT Is styled text

OBJECT Is styled text ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時:objectはオブジェクト名(文字列)省略時:objectは変数
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数(* 省略時)
戻り値	ブール	←	オブジェクトがマルチスタイルテキストであれば True 、そうでなければ False

説明

OBJECT Is styled textコマンドは、object と * によって指定されたオブジェクトの"マルチスタイル"オプションにチェックが入っているときに **True** を返します。

「マルチスタイル」オプションは、複数のスタイルバリエーションを一つのエリアで使用できるリッチテキストを使用可能にするものです。詳細な情報に関しては、*Design Reference* マニュアル内の マルチスタイル (リッチテキストエリア) を参照して下さい。

マルチスタイルのオブジェクトは、"" テーマ内のコマンドを使うことによってプログラムで管理することができます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

例題

あるフォーム内に、二つの異なるオブジェクトによって表現されたフィールドがあり、片方のオブジェクトは「マルチスタイル」のプロパティにチェックがされていて、もう片方にはチェックがされていません。このとき、以下の様なコードでこれを判別することが出来ます。

```
$Style:=OBJECT Is styled text(*;"Styled_text")
 // True が返されます （「マルチスタイル」がチェックされています）

$Style:=OBJECT Is styled text(*;"Plain_text")
 // False が返されます（「マルチスタイル」はチェックされていません）
```

OBJECT MOVE

OBJECT MOVE ({* ;} object ; moveH ; moveV {; resizeH {; resizeV {; *}}})

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
moveH	倍長整数	→	オブジェクトの水平移動量 (>0 = 右方向, <0 =左方向)
moveV	倍長整数	→	オブジェクトの垂直移動量 (>0 = 下方向, <0 = 上方向)
resizeH	倍長整数	→	オブジェクトの水平方向へのサイズ変更値
resizeV	倍長整数	→	オブジェクトの垂直方向へのサイズ変更値
*	演算子	→	指定時 = 絶対座標 省略時 = 相対座標

説明

OBJECT MOVEコマンドは、* とobject引数で指定されたカレントフォーム内のオブジェクトを、水平方向にmoveHピクセル、垂直方向にmoveVピクセル移動させます。
またオプションで、オブジェクトを水平方向にresizeHピクセル、垂直方向にresizeVピクセル、サイズの変更をすることもできます。

移動とサイズ変更の方向は、水平移動および垂直移動引数に渡された値に依ります:

- 値が正であれば、オブジェクトは右および下へそれぞれ移動され、サイズ変更されます。
- 値が負であれば、オブジェクトは左および上へそれぞれ移動され、サイズ変更されます。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。

objectにオブジェクト名としてワイルドカード文字 ("@") を使用し、複数のオブジェクトを指定すると、関連する全オブジェクトを移動またはサイズ変更することができます。

Note: バージョン6.5からは、文字列に含まれるワイルドカード文字 ("@") の取り扱いを設定することができます。このオプションは"オブジェクトプロパティ"コマンドに影響を与えます。詳細は4D Design Referenceを参照してください。

デフォルトではmoveH、moveV、resizeH、resizeVの値は、オブジェクトの以前の位置からの相対的な座標を変更します。引数が絶対位置を表わすようにしたい場合は、最後のオプションの引数 * を渡します。

このコマンドは以下のコンテキストで動作します:

- 入力フォームのデータ入力
- DIALOGコマンドを使用して表示されたフォーム
- MODIFY SELECTIONやDISPLAY SELECTIONコマンドで表示される出力フォームのヘッダとフッタ
- フォーム印刷イベント

例題 1

下記のコードは、"button_1"を右に10ピクセル、上に20ピクセル移動させ、幅を30ピクセル、高さを40ピクセルサイズ変更します:

```
OBJECT MOVE(*;"button_1";10;-20;30;40)
```

例題 2

下記のコード=は、"button_1"を座標(10;20)(30;40)に移動します:

```
OBJECT MOVE(*;"button_1";10;20;30;40;*)
```

OBJECT SET ACTION

OBJECT SET ACTION ({* ;} object ; action)

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
action	テキスト	⇒	関連付けるアクション

説明

OBJECT SET ACTIONコマンドは、引数 objectと * で指定したオブジェクトと関連付けられた標準アクションを、カレントプロセスにおいて変更します。

注: 標準アクションはデザインモードからプロパティリストを使用することによって設定することも可能です。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

action 引数には、オブジェクトと関連付けたい標準アクションを表すコードを文字列で渡します。"関連付けられた標準アクションのテキスト値" テーマ内にある、以下の定数のどれか一つを渡すことができます。

定数	型	値
Object Accept action	文字列	2
Object Add subrecord action	文字列	14
Object Automatic splitter action	文字列	16
Object Cancel action	文字列	1
Object Clear action	文字列	21
Object Copy action	文字列	19
Object Cut action	文字列	18
Object Database Settings action	文字列	32
Object Delete record action	文字列	7
Object Delete subrecord action	文字列	13
Object Edit subrecord action	文字列	12
Object First page action	文字列	10
Object First record action	文字列	5
Object Goto page action	文字列	15
Object Last page action	文字列	11
Object Last record action	文字列	6
Object MSC action	文字列	36
Object Next page action	文字列	8
Object Next record action	文字列	3
Object No standard action	文字列	0
Object Open back URL action	文字列	37
Object Open next URL action	文字列	38
Object Paste action	文字列	20
Object Previous page action	文字列	9
Object Previous record action	文字列	4
Object Quit action	文字列	27
Object Redo action	文字列	31
Object Refresh current URL action	文字列	39
Object Return to Design mode action	文字列	35
Object Select all action	文字列	22
Object Show Clipboard action	文字列	23
Object Stop loading URL action	文字列	40
Object Test Application action	文字列	26
Object Undo action	文字列	17

例題

ボタンの標準アクションをOKに設定したい場合、以下のように記述します:

```
OBJECT SET ACTION(*;"bValidate";Object Accept action)
```

OBJECT SET AUTO SPELLCHECK

OBJECT SET AUTO SPELLCHECK ({* ;} object ; autoSpellcheck)

引数	型		説明
*	演算子	⇨	指定時objectはオブジェクト名 (文字列) 省略時objectは変数またはフィールド
object	フォームオブジェクト	⇨	オブジェクト名 (* 指定時) 変数またはフィールド (* 省略時)
autoSpellcheck	ブール	⇨	True = 自動スペルチェック False = 自動スペルチェックなし

説明

OBJECT SET AUTO SPELLCHECKコマンドはobjectと*引数で指定されたオブジェクトの自動スペルチェックオプションをカレントプロセス内で動的に設定します。このオプションを使用して、オブジェクト (テキスト型オブジェクトのみ) にデータが入力される際の自動スペルチェックを有効/無効にできます。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

この機能を有効にするにはautoSpellcheckに**True**を、無効にするには**False**を渡します。

OBJECT SET BORDER STYLE

OBJECT SET BORDER STYLE (* ; object ; borderStyle)

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
borderStyle	倍長整数	→	境界線スタイル

説明

OBJECT SET BORDER STYLEコマンドは、引数 objectと * で指定したオブジェクトの境界線スタイルを変更します。

「境界線スタイル」のプロパティは、オブジェクトの外枠の見た目を管理します。詳細な情報に関しては *Design Reference* マニュアルの 境界線スタイル を参照して下さい。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

borderStyle 引数には、オブジェクトに適用したい境界線スタイルを指定する値を渡します。"Form Objects (Properties)" テーマ内にある、以下の定数のどれかを使用することができます:

定数	型	値	コメント
Border Dotted	倍長整数	2	オブジェクトの境界線は1ptの点線になります。
Border Double	倍長整数	5	オブジェクトの境界線は二重線(1ピクセル離れた2本の1ptの線)になります。
Border None	倍長整数	0	オブジェクトは境界線を持ちません。
Border Plain	倍長整数	1	オブジェクトの境界線は連続した一本の1ptの線になります。
Border Raised	倍長整数	3	オブジェクトの境界線は浮き上がったような3Dエフェクトになります。
Border Sunken	倍長整数	4	オブジェクトの境界線は沈み込んだような3Dエフェクトになります。
Border System	倍長整数	6	オブジェクトの境界線はシステムのグラフィック仕様に沿ったものになります。

OBJECT SET COLOR

OBJECT SET COLOR ({* ;} object ; color {; altColor})

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フィールド, 変数	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
color	倍長整数	→	オブジェクトの新しいカラー
altColor	倍長整数	→	リストボックスの奇数行カラー

説明

OBJECT SET COLOR コマンドは、objectで指定されたフォームオブジェクトの描画色と背景色を設定します。objectがリストボックスである場合は、追加の引数を使用して奇数行の描画色と背景色を設定します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。オブジェクト名に関する詳細はオブジェクトプロパティの節を参照してください。

altColorを使用し、リストボックスやリストボックスの列の奇数行に色を指定することができます。この引数を渡すと、color引数は偶数行にのみ適用されます。奇数行背景色を使用すると、リストが読みやすくなります。
objectがリストボックスオブジェクトを指す場合、奇数行背景色はリストボックス全体に適用されます。objectが列を指す場合、その列のみがカラー設定を使用します。

color (およびaltColor) は、文字色と背景色を設定します。カラーは以下の計算式で求められます:

カラー:=-(文字色+(256 * 背景色))

文字色と**背景色**はカラーパレット内のカラー番号 (0から255まで) です。**カラー**は常に負の数値です。例えば文字色が20で、背景色が10の場合、**カラー**は - (20 + (256 * 10)) つまり-2580となります。

Note: フォームエディタのプロパティリストウィンドウでカラーパレットを見ることができます。

一般的に使用されるカラーの番号はテーマにある次の定義済み定数により提供されています:

定数	型	値
Black	倍長整数	15
Blue	倍長整数	6
Brown	倍長整数	13
Dark Blue	倍長整数	5
Dark Brown	倍長整数	10
Dark Green	倍長整数	9
Dark Grey	倍長整数	11
Green	倍長整数	8
Grey	倍長整数	14
Light Blue	倍長整数	7
Light Grey	倍長整数	12
Orange	倍長整数	2
Purple	倍長整数	4
Red	倍長整数	3
White	倍長整数	0
Yellow	倍長整数	1

Note: OBJECT SET COLORはデフォルト4Dカラーパレットのインデックス化されたカラーを使用し、コマンド OBJECT SET RGB COLORSはRGBカラーを使用します。オブジェクトの自動カラーを再設定するにはOBJECT SET RGB COLORS コマンドでDefault foreground colorとDefault background color 定数を使用します。

例題

以下の例題ではフォームエディタで以下のように表示されるテキストエリアのカラーを設定します:

Information

以下のコードを実行後は:

```
OBJECT SET COLOR(*;"Mytext";-(Yellow+(256*Red)))
```

エリアは以下のように表示されます:

Information

OBJECT SET CONTEXT MENU

OBJECT SET CONTEXT MENU ({* ;} object ; contextMenu)

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
contextMenu	ブール	→	True = コンテキストメニュー有効、False = コンテキストメニュー無効

説明

OBJECT SET CONTEXT MENUコマンドは、引数 objectと * で指定したオブジェクトのデフォルトでのコンテキストメニューの関連付けを、カレントプロセスにおいて有効化または無効化します。

コンテキストメニューは、テキスト型の入力エリア、Web エリアとピクチャーに存在するオプションです。標準的なアクションメニューをこれらのオブジェクトに対して型に応じて関連付けすることができます。たとえばテキストオブジェクトに対してコピー・ペーストが使用できるかどうかなどです。詳細に関しては、*Design Reference* マニュアルを参照して下さい。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

contextMenu 引数には、コンテキストメニューを有効にしたい場合には **True** を、無効にしたい場合には **False** を渡します。

OBJECT SET COORDINATES

OBJECT SET COORDINATES ({* ;} object ; left ; top {; right ; bottom})

引数	型		説明
*	演算子	→	指定時:object はオブジェクト名(文字列)、省略時:object は変数またはフィールド
object	倍長整数	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
left	倍長整数	→	オブジェクトの左端の絶対座標(ピクセル)
top	倍長整数	→	オブジェクトの上端の絶対座標(ピクセル)
right	倍長整数	→	オブジェクトの右端の絶対座標(ピクセル)
bottom	倍長整数	→	オブジェクトの下端の絶対座標(ピクセル)

説明

OBJECT SET COORDINATESコマンドは、引数 objectと * で指定したオブジェクトの位置とそれに伴うサイズを、カレントのプロセスにおいて変更します。

注: このコマンドは、OBJECT MOVE コマンドに第2引数の * を渡した時と同じ動作をします。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

left と top 引数には、フォーム内の、object 引数で指定したオブジェクトの、絶対座標を渡します。

right と bottom 引数にも絶対座標を渡すことで、オブジェクトの右下隅の位置を指定することもできます。この隅の位置が left と top 引数で指定された隅に矛盾する場合、オブジェクトは自動的にリサイズされます。

注: オブジェクトのサイズを変更せずに位置だけを変更したい場合は、既存の OBJECT MOVE コマンドの使用が推奨されます。

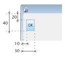
このコマンドは以下の様な場合においてのみ機能します:

- レコード編集中の入力フォーム
- DIALOG コマンドを使用して表示したフォーム
- MODIFY SELECTION か DISPLAY SELECTION フォームを使用して表示された出力フォームのヘッダーとフッター
- 印刷中のフォーム

例題

以下の宣言は、"button_1" のオブジェクトを、(10,20) (30,40) の座標に表示します:

```
OBJECT SET COORDINATES(*;"button_1";10;20;30;40)
```



OBJECT SET DATA SOURCE

OBJECT SET DATA SOURCE ({* ;} object ; dataSource)

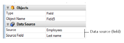
引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
dataSource	ポインター	→	オブジェクトの新しいデータソースへのポインター

説明

OBJECT SET DATA SOURCEコマンドは、引数 object と * で指定したオブジェクトのデータソースを変更します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

データソースとは、フォームを実行した際に、オブジェクトに表示される値を持っているフィールドまたは変数のことです。デザインモードでは、通常データソースはプロパティリスト内でソースとソースフィールド(フィールド)または変数名(変数)として定義されています:

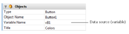


Objects

Type	Field
Object Name	Field5

Data Source

Source	Employees
Source Field	Last name



Objects

Type	Button
Object Name	Button1
Variable Name	vB1
Title	Colors

リストボックスを除き(後述)、フォーム内のすべてのデータソースはこのコマンドで変更することができます。データソースを変更することによって、アプリケーションの一貫性が損なわれないように気を付けてください。

リストボックスの場合は、以下の点に注意して下さい:

- データソースを変更する場合はリストボックスの型に注意する必要があります。たとえば、配列型のリストボックスの列のデータソースとして、フィールドを指定することはできない、ということです。
- セレクション型のリストボックスの場合、リストボックスオブジェクトそのもののデータソースを読み取ったり変更したりはできません。セレクション型のリストボックスの変数は、データソースではなくて内部参照として扱っているからです。
- このコマンドは主に配列型のリストボックスに対して使用されるものです。セレクション型のリストボックスに対しては、このコマンドの代わりに LISTBOX SET COLUMN FORMULA コマンドを使用することができます。

変更不可能なデータソースに対してこのコマンドが使用された場合、何も起こりません。

例題

入力エリアに対してデータソースを変更する場合:

```
C_POINTER($ptrField)
$ptrField:=Field(3;2)
OBJECT SET DATA SOURCE(*;"Input";$ptrField)
```

OBJECT SET DRAG AND DROP OPTIONS

OBJECT SET DRAG AND DROP OPTIONS ({* ;} object ; draggable ; automaticDrag ; droppable ; automaticDrop)

引数	型		説明
*	演算子	⇨	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	⇨	オブジェクト名 (* 指定時) または変数 (* 省略時)
draggable	ブール	⇨	ドラッグ可能ならTrue、そうでなければFalse
automaticDrag	ブール	⇨	自動ドラッグならTrue、そうでなければFalse
droppable	ブール	⇨	ドロップ可能ならTrue、そうでなければFalse
automaticDrop	ブール	⇨	自動ドロップ可能ならTrue、そうでなければFalse

説明

OBJECT SET DRAG AND DROP OPTIONSコマンドはobjectと * 引数で指定したオブジェクトのドラッグ＆ドロップオプションをカレントプロセス内で動的に設定します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

各引数にはそれぞれのオプションを有効にするか無効にするかを示すブール値を渡します:

- draggable = True: プログラムモードでオブジェクトのドラッグが可能
- automaticDrag = True (テキストフィールドや変数、コンボボックス、リストボックスでのみ使用): 自動モードでオブジェクトのドラッグが可能
- droppable = True: プログラムモードでオブジェクトのドロップが可能
- automaticDrop = True (テキストフィールドや変数、コンボボックス、リストボックスでのみ使用): 自動モードでオブジェクトのドロップが可能

例題

テキストエリアの自動ドラッグ＆ドロップを設定します:

```
OBJECT SET DRAG AND DROP OPTIONS(*;"Comments";False;True;False;True)
```

OBJECT SET ENABLED

OBJECT SET ENABLED ({* ;} object ; active)

引数	型		説明
*	演算子	⇒	指定時: objectはオブジェクト名 (文字) 省略時: objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または 変数 (* 省略時)
active	ブール	⇒	True = オブジェクトは有効; そうでなければFalse

説明

OBJECT SET ENABLED コマンドはobjectで指定されたカレントフォーム中のオブジェクトあるいはオブジェクトグループを、有効または無効にするために使用します。有効なオブジェクトはマウスクリックやキーボードショートカットに反応します。

オプションの* 引数を渡すと、object引数はオブジェクト名 (文字) です。この引数を渡さないと、objectは変数です。この場合、文字ではなく変数への参照 (変数オブジェクトのみ) を渡します。

オブジェクトを有効にするにはTrueを、無効にするにはFalseをactive引数に渡します。

このコマンドは以下のタイプのオブジェクトに適用できます:

- ボタン、デフォルトボタン、3Dボタン、非表示ボタン、ハイライトボタン
- ラジオボタン、3Dラジオボタン、ピクチャボタン
- チェックボックス、3Dチェックボックス
- ポップアップメニュー、ドロップダウンリスト、コンボボックス、メニュー/ドロップダウンリスト
- サーモメーター、ルーラー

Note: このコマンドは**入力**と**キャンセル**アクションを除き、標準アクションが割り当てられているボタンには効果がありません (4Dが必要に応じて状態を変更します)。

OBJECT SET ENTERABLE

OBJECT SET ENTERABLE ({* ;} object ; enterable)

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または テーブルまたはフィールドまたは変数 (* 省略時)
enterable	ブール	→	True: 入力可; False: 入力不可

説明

OBJECT SET ENTERABLE コマンドは、objectで指定したフォームオブジェクトを入力可または入力不可に設定します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。オブジェクト名に関する詳細はを参照してください。

このコマンドを使用することは、フォームエディタのプロパティリストウィンドウでフィールドや変数に対し入力可を設定することと同じです。このコマンドは、サブフォームのフォームメソッド内で使用されている場合にのみ、そのサブフォーム内で機能します。

entryArea入力可能 (True) であれば、ユーザはそのエリアにカーソルを移動してデータを入力することができます。entryAreaが入力不可 (False) の場合、ユーザはそのエリアにカーソルを移動してデータを入力することはできません。

また、サブフォームとMODIFY SELECTIONやDISPLAY SELECTIONコマンドを用いて表示されたリストフォームに対して、プログラムからリスト更新可モードを有効にするために、OBJECT SET ENTERABLEコマンドを使用することもできます:

- サブフォームの場合、entryArea引数にサブフォームテーブル名またはサブフォームオブジェクト名を渡すことができます。例: OBJECT SET ENTERABLE(*;"Subform";True)
- リストフォームの場合、entryArea 引数にはフォームのテーブル名を渡さなくてはなりません。例: OBJECT SET ENTERABLE([MyTable];True)

オブジェクトを入力不可にしても、プログラムから値を変更することはできます。

例題 1

以下の例は、船積みの重量に応じて、船積みフィールドを設定します。船積みが1オンス以下の場合は、shipperに米国郵便を設定しこのフィールドを入力不可にします。それ以外の場合には、入力可に設定します。

```
If([Shipments]Weight<=1)
   [Shipments]Shipper:="US Mail"
   OBJECT SET ENTERABLE([Shipments]Shipper;False)
Else
   OBJECT SET ENTERABLE([Shipments]Shipper;True)
End if
```

例題 2

次の例は、リストのヘッダに配置されたチェックボックスのオブジェクトメソッドであり、リスト更新モードを制御します:

```
C_BOOLEAN(bEnterable)
OBJECT SET ENTERABLE([Table1];bEnterable)
```

OBJECT SET EVENTS

OBJECT SET EVENTS ({* ;} object ; arrEvents ; mode)

引数	型		説明
*	演算子	→	指定時:object はオブジェクト名(文字列)、省略時:object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名または""(* 指定時)、または変数やフィールド(* 省略時)
arrEvents	倍長整数配列	→	設定したいイベントの配列
mode	倍長整数	→	arrEvents 引数で定義されたイベントの起動モード

説明

OBJECT SET EVENTSコマンドは、引数 objectと * で指定したフォームまたはオブジェクトのフォームイベントの設定を、カレントのプロセスにおいて変更します。

任意の* 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

フォーム自身のイベントの設定を定義する場合は、任意の * 演算子を渡したうえで object 引数に空の文字列 "" を渡します。こうすることでカレントフォームを指定します。

注: もしテーブルに関連したサブフォームのイベントを変更したい場合にはオブジェクト名を指定する記述法のみ有効です。

arrEvents 引数には、変更したい定義済みもしくはカスタムのフォームイベントのリストを倍長整数配列で渡します。(このとき mode 引数を使って、指定したイベントを有効にするか無効にするかを指定できます。)定義済みのイベントを指定するには、arrEvents 配列の要素の中に以下の定数を渡します。これらの定数は "Form Events" のテーマ内にあります。

定数	型	値	コメント
On Activate	倍長整数	11	フォームウィンドウが最前面のウィンドウになった
On After Edit	倍長整数	45	フォーカスのあるオブジェクトの内容が更新された
On After Keystroke	倍長整数	28	フォーカスのあるオブジェクトに文字が入力されようとしている。Get edited textはこの文字を含むオブジェクトのテキストを返す
On After Sort	倍長整数	30	(リストボックスのみ) リストボックスの列中で標準のソートが行われた
On Arrow Click	倍長整数	38	(3Dボタンのみ)3D ボタンの"三角"エリアがクリックされた
On Before Data Entry	倍長整数	41	(リストボックスのみ) リストボックスセルが編集モードに変更されようとしている
On Before Keystroke	倍長整数	17	フォーカスのあるオブジェクトに文字が入力されようとしている。Get edited textはこの文字を含まないオブジェクトのテキストを返す
On Begin Drag Over	倍長整数	46	オブジェクトがドラッグされている
On Begin URL Loading	倍長整数	47	(Webエリアのみ) 新しいURLがWeb エリアにロードされた
On bound variable change	倍長整数	54	サブフォームにバインドされた変数が更新された
On Clicked	倍長整数	4	オブジェクト上でクリックされた
On Close Box	倍長整数	22	ウィンドウのクローズボックスがクリックされた
On Close Detail	倍長整数	26	入力フォームから離れ、出力フォームに移動しようとしている
On Collapse	倍長整数	44	(階層リストまたは階層リストボックスのみ) クリックやキーストロークで階層リストの要素が折りたたまれた
On Column Moved	倍長整数	32	(リストボックスのみ) リストボックスの列がユーザのドラッグ＆ドロップで移動された
On Column Resize	倍長整数	33	(リストボックスのみ) リストボックスの列幅が変更された
On Data Change	倍長整数	20	オブジェクトのデータが変更された
On Deactivate	倍長整数	12	フォームウィンドウが最前面のウィンドウでなくなった
On Delete Action	倍長整数	58	(階層リストとリストボックスのみ) ユーザーが項目の削除を試みた
On Display Detail	倍長整数	8	レコードがリスト中に、あるいは行がリストボックス中に表示されようとしている
On Double Clicked	倍長整数	13	オブジェクト上でダブルクリックされた
On Drag Over	倍長整数	21	データがオブジェクト上にドロップされる可能性がある
On Drop	倍長整数	16	データがオブジェクトにドロップされた
On End URL Loading	倍長整数	49	(Webエリアのみ) URLのすべてのリソースがロードされた
On Expand	倍長整数	43	(階層リストまたは階層リストボックスのみ) クリックやキーストロークで階層リストの要素が展開された
On Footer Click	倍長整数	57	(リストボックスのみ) リストボックスあるいはリストボックス列でフッターがクリックされた
On Getting Focus	倍長整数	15	フォームオブジェクトがフォーカスを得た
On Header	倍長整数	5	フォームのヘッダエリアが印刷あるいは表示されようとしている
On Header Click	倍長整数	42	(リストボックスのみ) リストボックスの列ヘッダでクリックが行われた
On Load Record	倍長整数	40	リスト更新中、更新中にレコードがロードされた (ユーザがレコード行をクリックし、フィールドが編集モードになった)
On Long Click	倍長整数	39	(3Dボタンのみ) 3D ボタンがクリックされ、特定の時間以上マウスボタンが押され続けている
On Losing Focus	倍長整数	14	フォームオブジェクトがフォーカスを失った
On Mac toolbar button	倍長整数	55	Mac OSにおいて、ユーザーがツールバー管理ボタンをクリックした
On Menu Selected	倍長整数	18	メニュー項目が選択された
On Mouse Enter	倍長整数	35	マウスカーソルがオブジェクトの描画エリア内に入った
On Mouse Leave	倍長整数	36	マウスカーソルがオブジェクトの描画エリアから出た
On Mouse Move	倍長整数	37	マウスカーソルがオブジェクトの描画エリア上で (最低1ピクセル) 動いたか、変更キー(Shift, Alt, Shift Lock)が押された イベントがオブジェクトに対してのみチェックされていた場合は、マウスカーソルがオブジェクトのグラフィックエリアの中にあった場合にのみイベントが生成されます。
On Open Detail	倍長整数	25	レコードがダブルクリックされて、入力フォームを開こうとしている
On Open External Link	倍長整数	52	(Webエリアのみ) 外部URLがブラウザで開かれた
On Outside Call	倍長整数	10	フォームがCALL PROCESSによる呼び出しを受けた
On Picture Scroll	倍長整数	59	マウスやキーボードを使用して、ユーザーがピクチャーフィールドや変数の内容をスクロールした。
On Plug in Area	倍長整数	19	外部オブジェクトのオブジェクトメソッドの実行がリクエストされた
On Printing Break	倍長整数	6	フォームのブレークエリアのひとつが印刷されようとしている
On Printing Detail	倍長整数	23	フォームの詳細エリアが印刷されようとしている
On Printing Footer	倍長整数	7	フォームのフッタエリアが印刷されようとしている
On Resize	倍長整数	29	フォームウィンドウがリサイズされた
On Row Moved	倍長整数	34	(リストボックスのみ) リストボックスの行がユーザのドラッグ＆ドロップで移動された
On Selection Change	倍長整数	31	• リストボックス: 現在の行や列の選択が変更された • リスト中のレコード: リストフォームまたはサブフォームにおいて、カレントレコードあるいはカレントセレクションの行選択が変更された • 階層リスト: リスト中の選択がクリックやキーストロークなどで変更された • 入力可フィールドや変数 (v12.x以降): クリックやキー押下により、選択されたテキストやカーソルの位置がエリア内で変更された
On Timer	倍長整数	27	SET TIMERコマンドで設定した時間が経過した
On Unload	倍長整数	24	フォームを閉じる、あるいは解放しようとしている
On URL Filtering	倍長整数	51	(Webエリアのみ) Web エリアがURLをブロックした
On URL Loading Error	倍長整数	50	(Webエリアのみ) URLをロード中にエラーが発生した
On URL Resource Loading	倍長整数	48	(Webエリアのみ) 新しいリソースがWeb エリアにロードされた
On Validate	倍長整数	3	レコードのデータ入力が受け入れられた
On Window Opening Denied	倍長整数	53	(Webエリアのみ) ポップアップウィンドウがブロックされた

このリストでは On Load イベントが含まれていないことに注意して下さい。コマンドを実行した際には既にイベントが発生されてしまっているため、ここでは定義されていません。

arrEvents 引数にはカスタムイベントに対応する値を渡すこともできます。この際には、負の値を使用することが推奨されます。(CALL SUBFORM CONTAINER コマンドのドキュメントを参照して下さい。)

mode 引数には、配列の要素に対する全体的な処理の適用方法を決める値を渡します。"Form Objects (Properties)" テーマ内にある、以下の定数のどれかを渡して指定して下さい。

定数	型	値	コメント
Disable events others unchanged	倍長整数	2	arrEvents に指定された全てのイベントは無効化され、他は何も変更されません。
Enable events disable others	倍長整数	0	arrEvents に指定された全てのイベントは有効になり、他は全て無効化されます。
Enable events others unchanged	倍長整数	1	arrEvents に指定された全てのイベントは有効になり、他は何も変更されません。

OBJECT SET EVENTS コマンドを使用する際、object 引数で指定したオブジェクトでサポートされていない型のイベントの設定をしようとした場合、このイベントは無視されます。

オブジェクトが **OBJECT SET EVENTS** コマンドを呼び出した後に複製された場合は、有効/無効の設定も複製先に引き継がれます。

例題 1

いくつかのリストボックスオブジェクトに対して3つのフォームイベントを有効にし、他を全て無効にする場合:

```
ARRAY LONGINT($MyEventsOnLB;3)
$MyEventsOnLB {1}:=On After Sort
$MyEventsOnLB {2}:=On Column Moved
$MyEventsOnLB {3}:=On Column Resize
OBJECT SET EVENTS(*;"MyLB@";$MyEventsOnLB;Enable events disable others)
  // 指定した3つのイベントを有効にし、他は全て無効
```

例題 2

いくつかのリストボックスオブジェクトに対して3つのフォームイベントを無効にし、他は何も変更をしない場合:

```
ARRAY LONGINT($MyEventsOnLB;3)
$MyEventsOnLB {1}:=On After Sort
$MyEventsOnLB {2}:=On Column Moved
$MyEventsOnLB {3}:=On Column Resize
OBJECT SET EVENTS(*;"MyLB@";$MyEventsOnLB;Disable events others unchanged)
  // 指定した3つのイベントのみ無効
```

例題 3

あるオブジェクトのフォームイベントを有効にし、他は何も変更をしない場合:

```
ARRAY LONGINT($MyEventsOnLB;1)
$MyEventsOnLB {1}:=On Column Moved
OBJECT SET EVENTS(*;"Col1";$MyEventsOnLB;Enable events others unchanged)
  // 指定したイベントのみ有効
```

例題 4

フォームのイベントを全て無効にする場合:

```
ARRAY LONGINT($MyFormEvents;0)
OBJECT SET EVENTS(*;"";$MyFormEvents;Enable events disable others)
  // 全てのイベントが無効
```

例題 5

フォームのイベントを一つだけ無効にし、他は何も変更しない場合:

```
ARRAY LONGINT($MyFormEvents;1)
$MyFormEvents{1}:=On Timer
OBJECT SET EVENTS(*;"";$MyFormEvents;Disable events others unchanged)
  // 指定した一つのイベントのみ無効
```

OBJECT SET FILTER

OBJECT SET FILTER ({* ;} object ; entryFilter)

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
entryFilter	文字	→	入力可エリアの新しい入力フィルタ

説明

OBJECT SET FILTER は、objectで指定されたオブジェクトの入力フィルタをentryFilterに設定します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。オブジェクト名に関する詳細はを参照してください。

OBJECT SET FILTER は、入力フォームおよびダイアログ用フォームに対して使用でき、デザインモードで入力フィルタを受け付けるフィールドや入力可変数に適用できます。

entryFilterに空の文字列を指定すると、オブジェクトのカレント入力フィルタを取り除きます。

Note: このコマンドは、サブフォームのリストフォームに配置されたフィールドには使用できません。

Note: ツールボックスであらかじめ定義した入力フィルタをentryFilterに使用するには、入力フィルタ名の前に縦棒(|)を挿入します。

例題 1

以下の例は、郵便番号フィールドに対する入力フィルタを設定します。住所が米国の場合は、米国の郵便番号フィルタを設定します。それ以外の場合は、任意の入力ができるように設定します:

```
If([Companies]Country="US") ` ZIPコードフォーマットにフィルターを設定
   OBJECT SET FILTER([Companies]ZIP Code;"&9#####")
Else ` アルファベットと数字を受け付け、小文字を大文字に変換
   OBJECT SET FILTER([Companies]ZIP Code;"~@")
End if
```

例題 2

この例題では、"a," "b," "c," そして"g"のみの入力を許可するよう設定します:

```
OBJECT SET FILTER([table]field ;"&"+Char(Double quote)+"a;b;c;g"+Char(Double quote)+"##")
```

Note: この例題では入力フィルタを &"a;b;c;g"## に設定しています。

OBJECT SET FOCUS RECTANGLE INVISIBLE

OBJECT SET FOCUS RECTANGLE INVISIBLE ({* ;} object ; invisible)

引数	型		説明
*	演算子	⇒	指定時objectはオブジェクト名 (文字列) 省略時objectは変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) 変数またはフィールド (* 省略時)
invisible	ブール	⇒	True = フォーカスの四角を隠す False = フォーカスの四角を表示する

説明

OBJECT SET FOCUS RECTANGLE INVISIBLEコマンドはobjectと * 引数で指定したオブジェクトのフォーカス四角のカレントプロセスの表示オプションを変更します。この設定は、フォームエディターのプロパティリストの入力可能オブジェクトで利用できる**フォーカスの四角を隠す**オプションに対応します。

注: このオプションはMac OSでのみ利用できます。Windowsでは効果がありません。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

invisible 引数に**True**を渡すとフォーカスの四角が隠され、**False**を渡すと表示されます。

OBJECT SET FONT

OBJECT SET FONT ({* ;} object ; font)

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
font	文字	→	フォント名またはフォント番号

説明

OBJECT SET FONT はobject で指定したフォームオブジェクトに、font に渡したフォント名またはフォント番号のフォントが表示されるよう設定します。font 引数には有効なフォント名を渡さなければなりません。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。

例題 1

以下の例はbOKという名前のボタンに対してフォントを設定します:

```
OBJECT SET FONT(bOK;"Arial")
```

例題 2

以下の例はオブジェクト名に"info"を含むすべてのオブジェクトのフォントを設定します:

```
OBJECT SET FONT(*;"@info@";"Times")
```

例題 3

以下の例題は特別な *%password* フォントをパスワード入力欄に割り当てます (文字は隠されます)。

```
OBJECT SET FONT([Users]Password;"%password")
```

互換性に関する注意: %passwordフォントはまだ現時点でサポートされていますが、その利用はもうお勧めできません。FILTER KEYSTROKEコマンドを使用してパスワードの入力を管理するほうがより適切です。

OBJECT SET FONT SIZE

OBJECT SET FONT SIZE ({* ;} object ; size)

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
size	倍長整数	→	フォントサイズ (ポイント)

説明

OBJECT SET FONT SIZE はobject で指定したフォームオブジェクトが、sizeに渡したフォントサイズを使用して表示されるよう設定します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。オブジェクト名に関する詳細はを参照してください。

sizeには1から255までの整数を指定できます。実際のフォントサイズが存在しない場合、文字は拡大/縮小されます。

フォームで指定されるオブジェクトの領域は、指定したフォントサイズを表示するのに十分な大きさがなければなりません。十分な大きさがない場合には、テキストが途中までしか表示されなかったり、全く表示されなくなります。

例題 1

以下の例は、vtInfo変数のフォントサイズを設定します:

```
OBJECT SET FONT SIZE(vtInfo;14)
```

例題 2

以下の例は、"hl"という名前で始まるすべてのフォームオブジェクトのフォントサイズを設定します:

```
OBJECT SET FONT SIZE(*;"hl@";14)
```

OBJECT SET FONT STYLE

OBJECT SET FONT STYLE ({* ;} object ; styles)

引数	型		説明
*	演算子	⇒	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
styles	倍長整数	⇒	フォントスタイル

説明

OBJECT SET FONT STYLE はobject で指定したフォームオブジェクトが、stylesに渡したフォントスタイルを使用して表示されるよう設定します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。オブジェクト名に関する詳細はの節を参照してください。

styles にはフォントスタイルを定義する定数の和を渡します。4Dは以下の定義済み定数を提供します:

定数	型	値
Bold	倍長整数	1
Italic	倍長整数	2
Plain	倍長整数	0
Underline	倍長整数	4

例題 1

以下の例はbAddNewボタンのスタイルを設定します。フォントスタイルは太字のイタリックになります:

```
OBJECT SET FONT STYLE(bAddNew;Bold+Italic)
```

例題 2

以下の例は、オブジェクト名が"vt"で始まるすべてのフォームオブジェクトのフォントスタイルをスタイルなしにします:

```
OBJECT SET FONT STYLE(*;"vt@";Plain)
```

OBJECT SET FORMAT

OBJECT SET FORMAT ({* ;} object ; displayFormat)

引数	型		説明
*	演算子	⇒	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
displayFormat	文字	⇒	オブジェクトに設定する表示フォーマット

説明

OBJECT SET FORMAT は、objectで指定したオブジェクトの表示フォーマットをdisplayFormatで渡したフォーマットに設定します。新しいフォーマットは現在の表示にのみ有効です。フォームには保存されません。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。オブジェクト名に関する詳細はを参照してください。

OBJECT SET FORMAT は入力および出力フォーム (表示または印刷) 両方で使用でき、(入力可/不可) フィールドや変数に適用できます。

オブジェクトのデータタイプに適応する表示フォーマットを使用しなければなりません。

ブール

ブールフィールドをフォーマットするには二つの方法があります:

- 一つの値をdisplayFormatに渡す。この場合、フィールドはチェックボックスとして表示され、指定した値がラベルになります。
- セミコロン (;) で区切った二つの値をdisplayFormatに渡す。この場合フィールドは2つのラジオボタンとして表示されます。

日付

日付フィールドや変数をフォーマットするには、Char(n)をdisplayFormatに渡します。このときnは4Dにより提供される以下の定義済み定数のうちいずれかです:

定数	型	値	コメント
Blank if null date	倍長整数	100	0の代わりに""
Date RFC 1123	倍長整数	10	Fri, 10 Sep 2010 13:07:20 GMT
Internal date abbreviated	倍長整数	6	Dec 29, 2006
Internal date long	倍長整数	5	December 29, 2006
Internal date short	倍長整数	7	2006/12/29
Internal date short special	倍長整数	4	06/12/29 (しかし 1986/12/29 または 2096/12/29)
ISO Date	倍長整数	8	2006-12-29T00:00:00
ISO Date GMT	倍長整数	9	2010-09-13T16:11:53Z
System date abbreviated	倍長整数	2	
System date long	倍長整数	3	
System date short	倍長整数	1	

Note: Blank if null は他の定数に加算されなければなりません。この定数は日付がヌル値の時、00/00/00ではなく空のエリアとして表示するよう4Dに指示します。

時間

時間フィールドや変数をフォーマットするには、Char(n)をdisplayFormatに渡します。このときnは4Dにより提供される以下の定義済み定数のうちいずれかです:

定数	型	値	コメント
Blank if null time	倍長整数	100	0の代わりに""
HH MM	倍長整数	2	01:02
HH MM AM PM	倍長整数	5	1:02 AM
HH MM SS	倍長整数	1	01:02:03
Hour Min	倍長整数	4	1時2分
Hour Min Sec	倍長整数	3	1時2分3秒
ISO Time	倍長整数	8	0000-00-00T01:02:03
Min Sec	倍長整数	7	62分3秒
MM SS	倍長整数	6	62:03
System time long	倍長整数	11	1:02:03 AM HNEC (Macのみ)
System time long abbreviated	倍長整数	10	1:02:03 AM (Macのみ)
System time short	倍長整数	9	01:02:03

Note: Blank if null は他の定数に加算されなければなりません。この定数は日付がヌル値の時、00:00:00ではなく空のエリアとして表示するよう4Dに指示します。

ピクチャ

ピクチャフィールドや変数をフォーマットするには、Char(n)をdisplayFormatに渡します。このときnは4Dにより提供される以下の定義済み定数のうちいずれかです:

定数	型	値
On Background	倍長整数	3
Replicated	倍長整数	7
Scaled to Fit	倍長整数	2
Scaled to fit prop centered	倍長整数	6
Scaled to fit proportional	倍長整数	5
Truncated Centered	倍長整数	1
Truncated non Centered	倍長整数	4

文字と数値

文字や数値のフィールドや変数をフォーマットするには、displayFormat 引数に直接フォーマットラベルを渡します。

表示フォーマットに関する詳細は4D Design Referenceマニュアルのやを参照してください。

Note: ツールボックスであらかじめ定義した表示フォーマットをdisplayFormatに使用するには、表示フォーマット名の前に縦棒(|)を挿入します。

ピクチャボタン

ピクチャボタンをフォーマットするには、displayFormat 引数に以下のシンタックスを使用した文字列を渡します:
cols;lines;picture;flags{;ticks}

- cols = ピクチャの列数
- lines = ピクチャの行数
- picture = 使用するピクチャ (ピクチャライブラリ、ピクチャ変数、PICTリソース):
    - ピクチャライブラリのピクチャを使用する場合、クエスチョンマークの後にその番号を指定します (例: "?250")。
    - ピクチャ変数ののピクチャを使用する場合、変数名を指定します。
    - PICTリソースのピクチャを使用する場合、コロンの後にその番号を指定します (例: ":62500")。
- flags = ピクチャボタンの表示モードと処理。この引数には以下の値を指定できます: 0, 1, 2, 16, 32, 64, 128。これらの値はそれぞれ表示モードと処理モードを表します。これらの値は累計することができます。例えばモード1と64を有効にするには、65をflags 引数に渡します。それぞれの値の意味は以下のとおりです:
    - flags = 0 (オプションなし)
      ユーザがピクチャをクリックすると、順番に次のピクチャを表示します。Shiftキーを押しながらクリックすると、前のピクチャを順に表示します。最後のピクチャに達すると、クリックしてもピクチャは変更されません。つまり最初のピクチャには戻りません。
    - flags = 1 (連続してスイッチ)
      前のオプションと同様ですが、ユーザがマウスを押したままにするとピクチャは連続して (アニメーションのように) 変更されます。最後のピクチャに到達すると、そこでピクチャの変更は停止します。
    - flags = 2 (最初のフレームにループバック)
      前のオプションと同様ですが、ピクチャが連続したループで表示される点が異なります。最後のピクチャに達っし、さらにクリックすると、最初のピクチャが表示されます。
    - flags = 16 (ロールオーバー時にスイッチ)
      ピ クチャボタンのコンテンツは、マウスカーソルがボタンの上に来たときに変更されます。マウスがボタンエリアから離れると、最初のピクチャに戻ります。この モードはマルチメディアアプリケーションやHTMLドキュメントでしばしば使用されます。ロールオーバー時に表示されるピクチャはサムネールテーブルの最 後のピクチャです。ただし最後のフレームを無効として使用オプション (128) を使用した場合は、最後の前のフレームがロールオーバー時に使用されます。
    - flags = 32 (リリース後に元に戻す)
      このモードは2つのピクチャで動作します。これはユーザがクリックしたときを除き、常に最初のピクチャを表示します。この場合2番目のピクチャがマウスク リックの間表示され、マウスがリリースされると一番目のピクチャに戻ります。このモードを使用すれば、アイドルとクリック状態を表示するアクションボタン を作成できます。このモードを3D効果を作成したり、アクションを表現するピクチャを表示するために使用できます。
    - flags = 64 (透過)
      バックグランドピクチャを透過させるために使用します。
    - flags = 128 (最後のフレームを無効として使用)
      このモードは、最後のサムネールフレームをボタンが無効時に表示させるために使用します。このモードが選択されているとき、4Dは最後のサムネールを、ボタ ンが無効にされているときに表示します。このモードが、モード0, 1 および 2とともに使用されていると、最後のサムネールは一連の表示には組み込まれません。子のサムネールはボタンが無効のときにのみ表示されます。
- ticks = "n チック毎に表示"モードを有効にし、それぞれのピクチャを表示する間隔を設定します。このオプション引数が渡されると、指定された速度でピクチャボタンの コンテンツが繰り返し表示されます。例えば"2;3;?16807;0;10"と指定すると、ピクチャボタンは10tickごとに異なるピクチャを表示し ます。このモードが有効の時は透過モード (64) のみが使用できます。

ピクチャポップアップメニュー

ピクチャポップアップメニューをフォーマットするには、displayFormat 引数に以下のシンタックスを使用した文字列を渡します:
cols;lines;picture;hMargin;vMargin;flags

- cols = ピクチャの列数
- lines = ピクチャの行数
- picture = 使用するピクチャ (ピクチャライブラリ、ピクチャ変数、PICTリソース):
    - ピクチャライブラリのピクチャを使用する場合、クエスチョンマークの後にその番号を指定します (例: "?250")。
    - ピクチャ変数ののピクチャを使用する場合、変数名を指定します。
    - PICTリソースのピクチャを使用する場合、コロンの後にその番号を指定します (例: ":62500")。
- hMargin = メニューの水平境界とピクチャの間のマージン (ピクセル)
- vMargin = メニューの垂直境界とピクチャの間のマージン (ピクセル)
- flags = ピクチャポップアップメニューの透過モード (0 または 64):
    - mode = 0: ピクチャポップアップメニューは透過でない
    - mode = 64: ピクチャポップアップメニューは透過

サーモメーターおよびルーラー

サーモメータやルーラーをフォーマットするには、displayFormat 引数に以下のシンタックスを使用した文字列を渡します:
min;max;unit;step;flags{;format{;display}}

- min = インジケータの最初の目盛り値
- max = インジケータの最後の目盛り値
- unit = インジケータの目盛りの間隔
- step = インジケータ中でカーソル移動の最小間隔
- flags = インジケータの表示モードと動作。この引数は0, 2, 3, 16, 32, 128を受け入れます。これらの値は128を除き、加算して複数のオプションを設定できます:
    - flags = 0: 単位を表示しない
    - flags = 2: インジケータの右または下に単位を表示
    - flags = 3: インジケータの左または上に単位を表示
    - flags = 16: 単位に隣接して目盛りを表示
    - flags = 32: ユーザがインジケータを調整している間、On Data Changeを実行する。この値が使用されない場合、On Data Changeはユーザがインジケータの調整を終了したときにのみ発生します。
    - flags = 128: "バーバーショップ" (連続したアニメーション) モードを有効にします。この値をほかの値と一緒に使用することはできません。このモードでは、他の引数は無視されます (display引数が渡された場合を除く)。このモードに関する詳細は、*Design Reference*マニュアルを参照してください。
- format = インジケータの目盛りの表示フォーマット
  インジケータオブジェクトのサイズが小さいため単位やメモリが正しく表示できない場合、それらは隠されます。
- display = 特別な表示オプション。サーモメーターの場合、この引数はflagsサブ引数が128のときにのみ考慮されます。
    - display = 0 (または省略時): 標準のルーラを表示 / "バーバーショップ"タイプの連続したアニメーションを表示。
    - display = 1 : ルーラーの"ステッパー"モードを有効にする / サーモメーターの"非同期進捗"モードを有効にする。これらのオプションに関する詳細は*Design Reference*マニュアルを参照してください。

ダイアル

ダイアルをフォーマットするには、displayFormat 引数に以下のシンタックスを使用した文字列を渡します:
min;max;unit;step{;flags}

- min = インジケータの最初の目盛り値
- max = インジケータの最後の目盛り値
- unit = インジケータの目盛りの間隔
- step = インジケータ中でカーソル移動の最小間隔
- flags = ダイアルの処理モード (オプション)。この引数は32整を受け入れます: ユーザがインジケータを調整している間、On Data Changeを実行する。この値が使用されない場合、On Data Changeはユーザがインジケータの調整を終了したときにのみ発生します。

ボタングリッド

ボタングリッドをフォーマットするには、displayFormat 引数に以下のシンタックスを使用した文字列を渡します:
cols;lines

- cols = グリッドの列数
- lines = グリッドの行数

Note: フォームオブジェクトの表示フォーマットに関する詳細は、4D Design Referenceマニュアルを参照してください。

3D buttons

3Dボタンをフォーマットするには、displayFormat 引数に以下のシンタックスを使用した文字列を渡します:
title;picture;background;titlePos;titleVisible;iconVisible;style;horMargin;vertMargin;
iconOffset;popupMenu

- title = ボタンタイトル。この値はテキストまたはリソース番号 (例: ":16800,1") で指定できます。
- picture = ボタンにリンクするピクチャ。ピクチャライブラリ、ピクチャ変数、PICTリソース、またはResourcesフォルダのピクチャを使用できます:
    - ピクチャライブラリのピクチャを使用する場合、クエスチョンマークの後にその番号を指定します (例: "?250")。
    - ピクチャ変数ののピクチャを使用する場合、変数名を指定します。
    - PICTリソースのピクチャを使用する場合、コロンの後にその番号を指定します (例: ":62500")。
    - データベースのResourcesフォルダのピクチャを使用する場合、"#{folder/}picturename" または "file:{folder/}picturename"タイプのURIを指定します。
- background = ボタンにリンクするバックグランドピクチャ (カスタムスタイル)。ピクチャライブラリ、ピクチャ変数、PICTリソース、Resourcesフォルダのファイル (上記参照) を使用できます。
- titlePos = ボタンタイトルの位置:
    - titlePos = 1: 左
    - titlePos = 2: 上
    - titlePos = 3: 右
    - titlePos = 4: 下
    - titlePos = 5: 中央
- titleVisible = タイトルの表示/非表示:
    - titleVisible = 0: タイトルを非表示
    - titleVisible = 1: タイトルを表示
- iconVisible = アイコンの表示/非表示:
    - iconVisible = 0 : アイコンを非表示
    - iconVisible = 1 : アイコンを表示
- style = ボタンスタイル。このオプションで指定したスタイルにより、バックグランドなど他のオプションが有効になるかどうかが決まります:
    - style = 0: なし
    - style = 1: バックグランドオフセット
    - style = 2: プッシュボタン
    - style = 3: ツールバーボタン
    - style = 4: カスタム
    - style = 5: サークル
    - style = 6: システムスクエア (小)
    - style = 7: Office XP
    - style = 8: ベベル
    - style = 9: 角の丸いベベル
- horMargin = 水平マージン。ボタン内部の左右マージン (アイコンやテキストが描画されないエリア) をピクセル単位で指定します。
- vertMargin = 垂直マージン。ボタン内部の上下マージン (アイコンやテキストが描画されないエリア) をピクセル単位で指定します。
- iconOffset = 右および下方向へのアイコンのシフト。ピクセル単位で指定されるこの値は、ボタンがクリックされた際のボタンアイコンの右下方向へのシフトを指定します (同じ値が両方向に使用されます)。
- popupMenu = ボタンへのポップアップメニューの関連付け:
    - popupMenu = 0: ポップアップメニューなし
    - popupMenu = 1: リンクしたポップアップメニュー
    - popupMenu = 2: 分離したポップアップメニュー

いくつかのオプションは、すべての3Dボタンで有効というわけではありません。また特定のオプションを設定したくない場合もあるでしょう。あるオプションを渡さないようにするには、対応する値を省略します。例えばtitleVisibleとvertMarginオプションを省略するには、以下のように書きます:

```
OBJECT SET FORMAT(myVar;"NiceButton;?256;:562;1;;1;4;5;;5;0")
```

例題 1

以下のコードは[Employee]DateHired フィールドをInternal date longにフォーマットします。

```
OBJECT SET FORMAT([Employee]DateHired;Char(Internal date long))
```

例題 2

以下のコードは [Company]ZIP Code フィールドのフォーマットを、フィールドデータ長に基づいて変更します:

```
If(Length([Company]ZIP Code)=9)
   OBJECT SET FORMAT([Company]ZIP Code;"#####-####")
Else
   OBJECT SET FORMAT([Company]ZIP Code;"#####")
End if
```

例題 3

以下のコードはブールフィールドのフォーマットを変更して、Married または Unmarriedが表示されるようにします:

```
OBJECT SET FORMAT([Employee]Marital Status;"Married;Unmarried")
```

例題 4

以下のコードはブールフィールドのフォーマットを変更して、"Classified"というラベルのチェックボックスが表示されるようにします:

```
SET FORMAT([Folder]Classification;"Classified")
```

例題 5

1行4列のサムネイルがあります。このサムネイルをピクチャボタンに使用します ("デフォルト", "クリック", "ロールローバー" そして "無効時")。ロールオーバー時にスイッチとリリース後に元に戻す、そして最後のフレームを無効として使用オプションを有効にします:

```
OBJECT SET FORMAT(*;"Picture Button";"4;1;?15000;176")
```

例題 6

サーモメータをバーバーショップモードにします:

```
OBJECT SET FORMAT($Mythermo;";;;;128")
$Mythermo:=1 `Start animation
```

OBJECT SET HELP TIP

OBJECT SET HELP TIP ({* ;} object ; helpTip)

引数	型		説明
*	演算子	⇒	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または変数 (* 省略時)
helpTip	テキスト	⇒	ヘルプメッセージの内容

説明

OBJECT SET HELP TIPコマンドはobject と* 引数で指定したオブジェクトに割り当てられたヘルプをカレントプロセス内で動的に変更します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

フォームが実行されると、マウスカーソルがフィールドやオブジェクト上に移動したとき、ヘルプTipを表示させることができます。ヘルプTipはデザインモードのフォームエディターやストラクチャーエディターでも設定できます。

メッセージとして表示する内容をhelpTip 引数に渡します。以下のいずれかを渡すことができます:

- 文字列。例えば "区切り文字として / を使用"等。
- ヘルプTipを表示しないようにするには空の文字列 ("")。

OBJECT SET HORIZONTAL ALIGNMENT

OBJECT SET HORIZONTAL ALIGNMENT ({* ;} object ; alignment)

引数	型		説明
*	演算子	⇒	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
alignment	倍長整数	⇒	整列コード

説明

OBJECT SET HORIZONTAL ALIGNMENTコマンドは、引数objectおよび * で指定したオブジェクトを整列します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。

Form Objects (Properties)テーマ内にある以下の定数の1つをalignment引数に渡します:

定数	型	値
Align center	倍長整数	3
Align default	倍長整数	1
Align left	倍長整数	2
Align right	倍長整数	4

整列を適用できるフォームオブジェクトは次の通りです:

- コンボボックス
- スタティックテキスト
- グループエリア
- ポップアップメニュー/ドロップダウンリスト
- フィールド
- 変数
- リストボックス
- リストボックス列
- リストボックスヘッダー
- リストボックスフッター

OBJECT SET INDICATOR TYPE

OBJECT SET INDICATOR TYPE ({* ;} object ; indicator)

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
indicator	倍長整数	⇒	インジケータータイプ

説明

OBJECT SET INDICATOR TYPEコマンドは、引数 objectと * で指定したサーモメーターの進捗インジケーターのタイプを、カレントプロセスにおいて変更します。

インジケータータイプは、サーモメーターの表示方法を定義します。詳細な情報に関しては、*Design Reference* マニュアルの インジケーター を参照して下さい。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

indicator 引数には、表示したいインジケーターの種類を渡します。"Form Objects (Properties)" のテーマ内にある、以下の定数の中から選ぶことが出来ます。

定数	型	値	コメント
Asynchronous progress bar	倍長整数	3	連続したアニメーションを表示する、回転型のインジケーター
Barber shop	倍長整数	2	連続したアニメーションを表示するバー
Progress bar	倍長整数	1	標準の進捗バー

OBJECT SET KEYBOARD LAYOUT

OBJECT SET KEYBOARD LAYOUT ({* ;} object ; languageCode)

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) 変数またはフィールド (* 省略時)
languageCode	文字	→	RFC3066 ISO639 そして ISO3166 言語コード "" = 変更しない

説明

OBJECT SET KEYBOARD LAYOUTコマンドはobject と 引数で指定したオブジェクトに割り当てられたキーボードレイアウトをカレントプロセス内で動的に変更します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

languageCode には使用する言語の (RFC3066、ISO639、そしてISO3166に基づく) コードを示す文字列を渡します。詳細はSET DATABASE LOCALIZATIONコマンドの説明を参照してください。

OBJECT SET LIST BY NAME

OBJECT SET LIST BY NAME ({* ;} object {; listType}; list)

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
listType	倍長整数	→	Type of list: Choice list, Required list or Excluded list
list	文字	→	選択リストとして使用するリストの名前 (デザインモードで定義)

説明

OBJECT SET LIST BY NAMEコマンドは、object引数で指定されたオブジェクトあるいはオブジェクトグループの選択リストを、listに渡したリスト (デザインモードのリストエディタで定義) に設定あるいは置き換えます。list 引数に名前を渡すリストは、リストエディターまたはデザインモードを使用して予め作成されている必要があります。

このコマンドは入力フォーム、ダイアログフォームにおいて、テキストが入力可能な変数またはフィールドに適用することが出来ます。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。

注: このコマンドはサブフォームリストフォームのフィールドには使用できません。

OBJECT SET LIST BY NAME コマンドは、object 引数と * によって指定されたオブジェクトに関連付けられた全ての型のリスト(選択リスト、指定リスト、除外リスト)を設定または置き換えることが出来るようになりました。そのためには、listType 引数に、"Form Objects (Properties)" テーマ内の以下の定数のいづれか一つを渡して下さい。:

定数	型	値	コメント
Choice list	倍長整数	0	選択できる値のリスト(プロパティリスト内の「選択リスト」)。(デフォルト)
Excluded list	倍長整数	2	入力できない値のリスト。(プロパティリスト内の「除外リスト」)
Required list	倍長整数	1	入力可能な値のリスト(プロパティリストの"指定リスト"オプション)。

この引数を省略した場合、自動的に 0 (Choice list) を渡したものとみなされます。

カレントプロセスにおいて、object, と関連付けられたリストとの関連付けを解除するためには、解除したい型のリストに対して list 引数に空の文字列("")を渡して下さい。

例題 1

以下の例は、Shipperフィールドに選択項目リストを設定します。船積みを夜間に行う場合、夜間に船積みすることができる船積み業者のリストを選択項目リストに設定します。それ以外の場合は通常の船積み業者に設定します:

```
If([Shipments]Overnight)
   OBJECT SET LIST BY NAME([Shipments]Shipper;"Fast Shippers")
Else
   OBJECT SET LIST BY NAME([Shipments]Shipper;"Normal Shippers")
End if
```

例題 2

"color_choice" というリストを、"DoorColor"という単純なポップアップ/ドロップダウンリストと関連付ける場合:

```
OBJECT SET LIST BY NAME(*;"DoorColor";Choice list;"color_choice")
  // この場合、第3引数 (定数) は省略することが出来ます。
```

例題 3

"color_choice" というリストを "WallColor" というコンボボックスと関連付けたい場合を考えます。コンボボックスは入力可能なので、"black"や"purple" と言った色が入力されないようにしたい場合、これらの色を "excl_colors" というリストに入れて以下の様に除外します:

```
OBJECT SET LIST BY NAME(*;"WallColor";Choice list;"color_choice")
OBJECT SET LIST BY NAME(*;"WallColor";Excluded list;"excl_colors")
```

例題 4

これらのリストの関連付けを解除したい場合:

```
  // 選択リストの解除
OBJECT SET LIST BY NAME(*;"DoorColor";Choice list;"")
  // 入力が許可されていない値 (除外リスト) の解除
OBJECT SET LIST BY NAME(*;"WallColor";Excluded list;"")
```

OBJECT SET LIST BY REFERENCE

OBJECT SET LIST BY REFERENCE ({* ;} object {; listType}; list)

引数	型		説明
*	演算子	➡	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	➡	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
listType	倍長整数	➡	リストの種類:選択リスト、指定リスト、除外リスト
list	ListRef	➡	リストの参照番号

説明

OBJECT SET LIST BY REFERENCEコマンドは、引数 objectと * で指定したオブジェクトと関連付けられたリストを、list 引数で指定した階層リストで置き換えもしくは定義します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

listType 引数を省略した場合、コマンドは自動的にソースとなる選択リストを定義します。listType 引数では設定したいリストのタイプを指定することができます。"Form Objects (Properties)" テーマ内にある以下の定数のどれかを渡して下さい:

定数	型	値	コメント
Choice list	倍長整数	0	選択できる値のリスト(プロパティリスト内の「選択リスト」)。(デフォルト)
Excluded list	倍長整数	2	入力できない値のリスト。(プロパティリスト内の「除外リスト」)
Required list	倍長整数	1	入力可能な値のリスト(プロパティリストの"指定リスト"オプション)。

list 引数には、オブジェクトに関連付けたい階層リストの参照番号を渡します。このリストは、Copy list コマンド、Load list コマンド、または New list コマンドを使用してすでに生成されている必要があります。

objectと list の関連付けを解除するためには、解除したいリストの種類に対し、list 引数に 0 を渡します。

リストの関連付けを解除しても、リスト参照はメモリーからは消去されません。リストが不要になった時には CLEAR LIST コマンドを使用して下さい。

このコマンドは、変数またはフィールドに関連付けられたポップアップまたはコンボボックスに対して使用すると興味深い挙動をします(*Design Reference* マニュアルを参照して下さい)。この場合、関連付けは動的に行われ、リストの変更は全てフォームへと反映されます。オブジェクトが配列に関連付けられたとき、リストは配列へとコピーされ、リストへのいかなる変更も自動的に不可になります(例5を参照して下さい)。

例題 1

単純な選択リスト(デフォルトのリスト型)をテキストフィールドへと関連付ける場合:

```
vCountriesList:=New list
APPEND TO LIST(vCountriesList;"Spain";1)
APPEND TO LIST(vCountriesList;"Portugal";2)
APPEND TO LIST(vCountriesList;"Greece";3)
OBJECT SET LIST BY REFERENCE([Contact]Country;vCountriesList)
```

例題 2

"vColor" というリストを単純な選択リストとして "DoorColor" ポップアップリストに関連付けます:

```
vColor:=New list
APPEND TO LIST(vColor;"Blue";1)
APPEND TO LIST(vColor;"Green";2)
APPEND TO LIST(vColor;"Red";3)
APPEND TO LIST(vColor;"Yellow";4)
OBJECT SET LIST BY REFERENCE(*;"DoorColor";Choice list;vColor)
```

例題 3

更に、"vColor" のリストを、"WallColor" というコンボボックスに関連付けする場合を考えます。このコンボボックスは入力可能なので、"black" "purple" と言った色が入力されないようにしたい場合、これらの色を "vReject" というリストに入れて除外します:

```
OBJECT SET LIST BY REFERENCE(*;"WallColor";Choice list;vColor)
vReject:=New list
APPEND TO LIST(vReject;"Black";1)
APPEND TO LIST(vReject;"Gray";2)
APPEND TO LIST(vReject;"Purple";3)
OBJECT SET LIST BY REFERENCE(*;"WallColor";Excluded list;vReject)
```

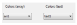
例題 4

これらのリストの関連付けを解除する場合:

```
OBJECT SET LIST BY REFERENCE(*;"WallColor";Choice list;0)
OBJECT SET LIST BY REFERENCE(*;"WallColor";Required list;0)
OBJECT SET LIST BY REFERENCE(*;"WallColor";Excluded list;0)
```

例題 5

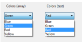
この例では、このコマンドがテキスト配列に関連付けられたポップアップメニューに適用されたときと、テキスト変数に関連付けられたポップアップメニューに適用されたときの違いについて説明します。フォーム内に二つのポップアップメニューがあるとします:



これらのポップアップメニューの中身は <>*vColor* list を使用して設定されています(中身はカラーの値です)。以下のコードがフォームをロードした際に実行されるとします:

```
ARRAY TEXT(arr1;0) //arr1 pop up
C_TEXT(text1) //text1 pop up
OBJECT SET LIST BY REFERENCE(*;"arr1";<>vColor)
OBJECT SET LIST BY REFERENCE(*;"text1";<>vColor)
```

メニューの中身はどちらも同一内容です:

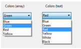


(上記の画像はメニューの中身を同時に表示しています)

次に、ボタンなどを使用して以下のコードを実行します:

```
APPEND TO LIST(<>vColor;"White";5)
APPEND TO LIST(<>vColor;"Black";6)
```

テキストフィールドと関連付けられていたメニューのみが(動的な参照によって)更新されます:



配列によって管理されたポップアップに関連付けられたリストを更新するためには、**OBJECT SET LIST BY REFERENCE** コマンドを再度使用してリストの中身をコピーする必要があります。

OBJECT SET MAXIMUM VALUE

OBJECT SET MAXIMUM VALUE ({* ;} object ; maxValue)

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
maxValue	日付, 時間, Number	→	オブジェクトの最大値

説明

OBJECT SET MAXIMUM VALUEコマンドは、引数 objectと * で指定したオブジェクトの最大値を、カレントプロセスにおいて変更します。

「最大値」プロパティは、数、日付、時間のデータ種類に対して適用することができます。詳細な情報に関しては、*Design Reference* マニュアルの最大値と最小値 を参照して下さい。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

maxValue 引数には、カレントプロセスにおいてオブジェクトに対して適用したい最大値を渡します。このとき、渡す最大値はオブジェクトの型と対応している必要があります。そうでない場合にはエラー18 "フィールドタイプが対応していません。"が返されます。

OBJECT SET MINIMUM VALUE

OBJECT SET MINIMUM VALUE ({* ;} object ; minValue)

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
minValue	日付, 時間, Number	⇒	オブジェクトの最小値

説明

OBJECT SET MINIMUM VALUEコマンドは、引数 objectと * で指定したオブジェクトの最小値を、カレントプロセスにおいて変更します。

「最小値」プロパティは、数、日付、時間のデータ種類に対して適用することができます。詳細な情報に関しては、*Design Reference* マニュアルの 最大値と最小値 を参照して下さい。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

minValue 引数には、カレントプロセスにおいてオブジェクトに対して適用したい最小値を渡します。このとき、渡す最小値はオブジェクトの型と対応している必要があります。そうでない場合にはエラー18"フィールドタイプが対応していません。"が返されます。

OBJECT SET MULTILINE

OBJECT SET MULTILINE ({* ;} object ; multiline)

引数	型		説明
*	演算子	→	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
multiline	倍長整数	→	複数行プロパティの状態

説明

OBJECT SET MULTILINEコマンドは、引数 objectと * で指定したオブジェクトの"複数行"のプロパティを変更します。

複 数行のプロパティはテキストエリアの表示と印刷に関係する二つの状態を管理します。単独行の最後に置かれる単語の表示と、改行の自動挿入です。詳細な情報 に関しては *Design Reference* マニュアルの 複数行 を参照して下さい。このプロパティが存在しないオブジェクトに対してコマンドを適用した場合、何も起こりません。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

multiline 引数には、設定を決める値を渡します。値としては "Form Objects (Properties)" テーマ内にある、以下の定数を使用することができます:

定数	型	値	コメント
Multiline Auto	倍長整数	0	単独行のエリアでは、行に表示しきれない単語は切り落とされ、改行はされません。 複数行のエリアでは、改行が行われます。
Multiline No	倍長整数	2	改行は禁止されます。テキストは必ず単独行として表示されます。文字列かテキストフィールドか変数に改行が含まれていたとしても、改行は行われません。
Multiline Yes	倍長整数	1	単独行のエリアでは、テキストは最初の改行までか、単語全体を表示できる最後の単語までが表示されます。その後改行が挿入されるので、↓キーを押すことによってエリアの内容をスクロールすることができます。 複数行のエリアでは、自動で改行が行われます。

例題

入力エリアにおいて、改行を禁止したい場合:

```
OBJECT SET MULTILINE(*;"vEntry";Multiline No)
```

OBJECT SET PLACEHOLDER

OBJECT SET PLACEHOLDER ({* ;} object ; placeholderText)

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
placeholderText	テキスト	⇒	オブジェクトに関連付けるプレースホルダーテキスト

説明

OBJECT SET PLACEHOLDERコマンドは、引数 objectと * で指定したオブジェクトにプレースホルダーテキストを関連付けます。

詳細な情報に関しては、*Design Reference* マニュアルを参照して下さい。

プロパティリストを通じてプレースホルダーが既に関連付けられていた場合、そのテキストはカレントプロセスにおいて placeholderText 引数で指定した値で置換されます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

placeholderText 引数には、オブジェクトが空欄の時に表示させたいヘルプテキストか指示を渡します。

注: OBJECT SET PLACEHOLDER コマンドでは、プレースホルダーに xliff 参照を挿入することはできません。挿入する場合にはプロパティリストを通じてプレースホルダーを定義して下さい。

このコマンドは、変数、フィールド、コンボボックスのフォームオブジェクトにしか使用できません。
プレースホルダーには文字列かテキストの値を関連付けることが出来ます。また、「ヌルのときブランクにする」プロパティがあるフォームオブジェクトに関しては、日付か時刻のデータと関連付けることができます。

例題

コンボボックスにプレースホルダーとして「Search」というテキストを表示したい場合:

```
OBJECT SET PLACEHOLDER(*;"search_combo";"Search")
```

OBJECT SET PRINT VARIABLE FRAME

OBJECT SET PRINT VARIABLE FRAME ({* ;} object ; variableFrame {; fixedSubform})

引数	型	説明
*	演算子	→ 指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	→ オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
variableFrame	ブール	→ True = 可変長フレーム印刷、False = 固定長フレーム印刷
fixedSubform	倍長整数	→ サブフォームを固定サイズで印刷する際のオプション

説明

OBJECT SET PRINT VARIABLE FRAMEコマンドは、引数 objectと * で指定したオブジェクトの印刷時可変のプロパティを、変更します。

このプロパティはテキスト型・ピクチャー型の変数とフィールド、そしてサブフォームにのみ存在します。サブフォームの場合は更に固定長印刷のオプションも存在します。詳細な情報に関しては、*Design Reference* マニュアルの 可変サイズエリアの印刷 を参照して下さい。このコマンドは、印刷時可変のプロパティがないオブジェクトに対しては何もしません。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

variableFrame 引数はブール型の引数です。**True**を渡した場合、オブジェクトは可変長フレームで印刷されます。**False** を渡した場合、オブジェクトは固定長フレームで印刷されます。

任意の fixedSubform 引数では、object でサブフォームを指定していて variableFrame 引数に **False** を渡した場合のみ、追加のオプションを設定できます(他の場合には無視されます)。このオプションではサブフォームの固定長フレーム印刷モードを定義できます。"Form Objects (Properties)" テーマにある、以下の定数のどちらかを渡して下さい:

定数	型	値	コメント
Print Frame fixed with multiple records	倍長整数	2	フレームは同じサイズを維持しますが、4Dは全てのレコードが載るまで複数回フォームを印刷します。
Print Frame fixed with truncation	倍長整数	1	4Dはサブフォームのエリアに収まるレコードのみ印刷します。フォームは一度だけ印刷され、印刷されなかったレコードは無視されます。

OBJECT SET RESIZING OPTIONS

OBJECT SET RESIZING OPTIONS ({* ;} object ; horizontal ; vertical)

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
horizontal	倍長整数	→	横リサイズオプション
vertical	倍長整数	→	縦リサイズオプション

説明

OBJECT SET RESIZING OPTIONSコマンドはobjectと* で指定したオブジェクトのリサイズオプションをカレントプロセス内で動的に変更します。これらのオプションを使用して、フォームウィンドウのサイズが変更されたときにオブジェクトをどのように表示するかを指定できます。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

horizontal 引数には横方向のリサイズオプションを渡します。Form Objects (Properties)テーマの以下の定数を使用できます:

定数	型	値	コメント
Resize horizontal grow	倍長整数	1	ウィンドウが横方向に拡げられたら、オブジェクトの幅も同じ比率だけ右に拡大する。
Resize horizontal move	倍長整数	2	ウィンドウの幅が拡げられたら、オブジェクトも同じだけ右方向に移動する。
Resize horizontal none	倍長整数	0	ウィンドウの幅が変更されても、オブジェクトの位置及びサイズを変更しない。

vertical 引数には縦方向のリサイズオプションを渡します。Form Objects (Properties)テーマの以下の定数を使用できます:

定数	型	値	コメント
Resize vertical grow	倍長整数	1	ウィンドウが縦方向に拡げられたら、オブジェクトの高さも同じ比率だけ下に拡大する。
Resize vertical move	倍長整数	2	ウィンドウの高さが拡げられたら、オブジェクトも同じだけ下方向に移動する。
Resize vertical none	倍長整数	0	ウィンドウの高さが変更されても、オブジェクトの位置及びサイズを変更しない。

OBJECT SET RGB COLORS

OBJECT SET RGB COLORS ({* ;} object ; foregroundColor ; backgroundColor {; altBackgrndColor})

引数	型		説明
*	演算子	⇨	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	⇨	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
foregroundColor	倍長整数	⇨	前景色のRGBカラー
backgroundColor	倍長整数	⇨	背景色のRGBカラー
altBackgrndColor	倍長整数	⇨	奇数行の背景色RGBカラー

説明

OBJECT SET RGB COLORS コマンドは、引数objectとオプション引数の * によって指定されるオブジェクトの前景色と背景色を変更します。コマンドがリストボックスに対して適用される場合、引数を更に使用して奇数行の背景色を変更することができます。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。オブジェクト名に関する詳細はを参照してください。

オプションの引数altBackgrndColorを使用して、奇数行の背景色を設定することができます。この引数は、指定されたオブジェクトがリストボックスまたはリストボックスの列である場合にのみ使用できます。この引数を使用した場合、backgroundColor引数は偶数行に対してのみ使用されます。奇数行背景色を使用すると、リストがより読みやすくなります。
object がリストボックスオブジェクトを指す場合、奇数行背景色はリストボックス全体に対して使用されます。object がリストボックスの任意の列を指す場合、設定した色はその列に対してだけ使用されます。

引数foregroundColor、backgroundColor、altBackgrndColorにはRGBのカラー値を指定します。RGBの値は、4バイトの倍長整数です。そのフォーマット(0x00RRGGBB)は、以下の表で説明します(バイトには、右から左へ順に0から3までの数字が付けられます):

Byte	説明
3	絶対RGBカラーの場合0でなければなりません。
2	カラーの赤コンポーネント (0..255)
1	カラーの緑コンポーネント (0..255)
0	カラーの青コンポーネント (0..255)

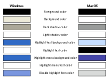
以下はRGBカラーの例です:

値	説明
0x00000000	黒
0x00FF0000	明るい赤
0x0000FF00	明るい緑
0x000000FF	明るい青
0x007F7F7F	グレイ
0x00FFFF00	明るい黄
0x00FF7F7F	パステル赤
0x00FFFFFF	白

カラーが自動で設定されるオブジェクトの描画を、4Dが使用する"システム"カラーで定義できます。以下の定義済み定数が提供されます:

定数	型	値	コメント
Background color	倍長整数	-2	
Background color none	倍長整数	-16	この定数は *backgroundColor* 引数と *altBackgrndColor* 引数にしか渡すことができません。
Dark shadow color	倍長整数	-3	
Disable highlight item color	倍長整数	-11	
Foreground color	倍長整数	-1	
Highlight menu background color	倍長整数	-9	
Highlight menu text color	倍長整数	-10	
Highlight text background color	倍長整数	-7	
Highlight text color	倍長整数	-8	
Light shadow color	倍長整数	-4	

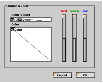
これらのカラーを以下に示します:


Windows
MacOS
Foreground color
Background color
Dark shadow color
Light shadow color
Highlight text background color
Highlight text color
Highlight menu background color
Highlight menu text color
Disable highlight item color

警告: これらの自動カラーはシステムに依存します。システムカラーが変更されると、4Dも変更された自動カラーを使用します。自動カラー値はシステムカラーを使用する際に指定してください。上記例のカラーを直接指定するためには使用できません。

例題 1

以下のフォームにはvsColorValueとvsColorという2つの入力不可変数とthRed、thGreen、thBlueという3つのサーモメータが含まれます。


Choose a Color
Color Value:
vsColorValue
Color:
vsColor
Red
Green
Blue
Cancel
OK

以下は、これらのオブジェクト用メソッドです:

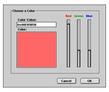
```
  // vsColorValue入力不可オブジェクトメソッド
 Case of
    :(Form event=On Load)
       vsColorValue:="0x00000000"
 End case
  // vsColor入力不可変数オブジェクトメソッド
 Case of
    :(Form event=On Load)
       vsColor:=""
       OBJECT SET RGB COLORS(vsColor;0x00FFFFFF;0x0000)
 End case

  // thRed サーモメーターオブジェクトメソッド
 CLICK IN COLOR THERMOMETER

  // thGreen サーモメーターオブジェクトメソッド
 CLICK IN COLOR THERMOMETER

  // thBlue サーモメーターオブジェクトメソッド
 CLICK IN COLOR THERMOMETER
```

3つのサーモメータから呼ばれるプロジェクトメソッドは 以下のとおり:

```
  // CLICK IN COLOR THERMOMETER プロジェクトメソッド
 OBJECT SET RGB COLORS(vsColor;0x00FFFFFF;(thRed<<16)+(thGreen<<8)+thBlue)
 vsColorValue:=String((thRed<<16)+(thGreen<<8)+thBlue;"&x")
 If(thRed=0)
    vsColorValue:=Substring(vsColorValue;1;2)+"0000"+Substring(vsColorValue;3)
 End if
```

サーモメータの値からカラー値を計算するためにを使用しています。

実行されるとフォームは 以下のように表示されます:


Choose a Color
Red
Green
Blue
Cancel
OK

Color Value: 0x00E05858

Color:

例題 2

背景色を透明に、フォントカラーを明るい色に設定する場合:

Simple Text

```
 OBJECT SET RGB COLORS(*;"myVar";Light shadow color;Background color none)
```

Simple Text

OBJECT SET SCROLL POSITION

OBJECT SET SCROLL POSITION ({* ;} object {; vPosition {; hPosition}}{; *})

引数	型		説明
*	演算子	➡	指定された場合、オブジェクトがオブジェクト名 (文字列) 省略された場合、オブジェクトがテーブルまたは変数
object	フォームオブジェクト	➡	オブジェクト名 (* が指定された場合) または テーブルまたは変数 (* が省略された場合)
vPosition	倍長整数	➡	表示する行番号、またはピクチャーの場合縦スクロール量 (ピクセル)
hPosition	倍長整数	➡	リストボックスの場合表示する列番号、またはピクチャーの場合縦スクロール量 (ピクセル)
*	演算子	➡	スクロール後の最初の位置に行を表示 (hPositionが渡された場合、列も対象)

説明

OBJECT SET SCROLL POSITIONコマンドを使用して、(MODIFY SELECTIONまたは[#cmdid="59"/]コマンドを用いて表示される) リストフォーム、サブフォーム、階層リストなどの行、あるいはリストボックスの列や行、そしてさらにピクチャのピクセルをスクロールすることができます。

注: プログラムによるスクロールは、フォーム上でスクロールバーが隠されていても可能です。

最初のオプション引数 * を渡すと、引数 object は、サブフォーム、階層リスト、リストボックス、ピクチャーフィールド/変数オブジェクトの名前であることを指示します (この場合、文字列を object に渡します) 。この引数に何も渡さないと、引数 object は、テーブル (リストフォームテーブルまたはサブフォームテーブル) または変数 (階層リストのListRef、リストボックス、またはピクチャ) であることを示します。

vPosition引数を使用して、表示する行番号を指定したり、ピクチャーの場合は適用する縦スクロール量を指定できます。
この引数を渡さないと、コマンドは選択されている最初の行が表示されるように縦スクロールを実行します。この場合、行が選択されていなかったり、選択された行が一つでも表示されていれば、縦スクロールは実行されません。
この引数を渡すと、コマンドは、設定された行が (ハイライトされているかいないかに関わらず)、指定された行が表示されるようにリストの縦スクロールを行います。行がすでに表示されていると、2番目の * 引数が渡されていない限り、コマンドはなにも行いません (後述)。

- リストフォームとサブフォームにおいて、この番号はカレントセレクション中の行番号 (位置) です。
- 階層リストの場合、コマンドは項目の展開/折りたたみ状態を考慮します。
- リストボックスの場合、この番号は (非表示行を含む) すべてのオブジェクト行中の行番号です。vPosition に渡された番号がリストボックスの非表示行に対応する場合、コマンドは続く最初の表示行を表示します。

  注: このコマンドは、たとえリストボックスが階層モードで表示されていたとしても、非階層の標準表示に基づいて実行される点に留意してください。リストボックスが階層モードあるいは標準モードどちらで表示されているかによって結果は異なります (例題参照)。

- フォームに表示されているピクチャーの場合、vPosition はピクチャーに適用される縦スクロール量を示します。縦スクロールを行わない場合、vPositionに0を渡します。値はローカルコンテキスト中でピクチャの基点に対し相対的なピクセルで表現しなければなりません (二番目の引数に * を渡した場合を除く、後述参照)。ピクチャは"トランケート (中央合わせなし)" フォーマットで表示されなければなりません。

hPosition 引数はリストボックスとピクチャーのコンテキストで使用できます。

- リストボックスの場合、hPositionに列番号を渡すことができます。コマンドを実行するとリストボックスを横スクロールし、指定した列が表示されます。列がすでに表示されている場合、コマンドはなにも行いません。縦スクロールと同様、2番目のオプション引数 * を渡すと、コマンドによって表示された列は先頭位置に配置されます (リストボックスが実際にスクロールされた場合。後述)。
- フォームに表示されたピクチャーの場合、hPosition はピクチャーに適用される横スクロール量を示します。値はローカルコンテキスト中でピクチャの基点に対し相対的なピクセルで表現しなければなりません (二番目の引数に * を渡した場合を除く、後述参照)。

二番目のオプション引数 * を渡すと:

- このコマンドで表示された行は (リストが実際にスクロールされると)、リスト中の先頭に配置されます。行がリストの最後の行である場合、このオプションは効果を持ちません。
- ピクチャーの場合、起点ではなく現在の位置から相対的にスクロールを行います。

注: HIGHLIGHT RECORDSコマンドはオプション引数 * を使用してスクロール管理を**OBJECT SET SCROLL POSITION**コマンドに委譲することができます。

例題

この例題ではリストボックスが標準モードで表示されている場合と階層モードで表示されている場合の違いについて説明します:

```
 OBJECT SET SCROLL POSITION(*;"mylistbox";4;2;*) // displays 4th row of 2nd column of list box in the first position
```

このステートメントが標準モードで表示されているリストボックスに適用されると:

France	Brittany	Brest	120000	...
France	Brittany	Quimper	80000	...
France	Brittany	Rennes	200000	...
France	**Normandy**	Caen	220000	...
France	Normandy	Deauville	4000	...
France	Normandy	Cherbourg	41000	...
...	...	...	...	...

リストボックスの行と列が実際にスクロールされます:

Normandy	Caen	220000	...	...
Normandy	Deauville	4000	...	...
Normandy	Cherbourg	41000	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...

他方同じステートメントが階層モードのリストボックスに適用されると、行はスクロールされますが、2番目の列は階層の一部なので列はスクロールされません:

V **Normandy**	
Caen	220000
Deauville	4000
Cherbourg	41000
...	

OBJECT SET SCROLLBAR

OBJECT SET SCROLLBAR ({* ;} object ; horizontal ; vertical)

引数	型		説明
*	演算子	⇒	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時), または 変数 (* 省略時)
horizontal	ブール	⇒	True = 表示, False = 非表示
vertical	ブール	⇒	True = 表示, False = 非表示

説明

OBJECT SET SCROLLBARコマンドは、引数objectと * で指定したオブジェクトの水平/垂直スクロールバーの表示/非表示を設定します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectは変数です。この場合、文字列ではなく変数参照を指定します。オブジェクト名に関する詳細はを参照してください。

このコマンドが使えるのは 以下のフォームオブジェクトです:

- リストボックス
- スクロールエリア
- 階層リスト
- サブフォーム

引数horizontalとverticalにブール値を指定し、スクロールバーの表示 (True) 、非表示 (False) を設定します。スクロールバーは、デフォルトでは表示されます。

注：スクロールエリアには、水平スクロールバーはありません。しかし、引数としてhorizontalを省略することができないため指定する必要があります。しかし、この値は意味を持たず、無視されます。

OBJECT SET SHORTCUT

OBJECT SET SHORTCUT ({* ;} object ; key {; modifiers})

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数またはフィールド
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) 変数またはフィールド (* 省略時)
key	文字	→	オブジェクトに割り当てるキー
modifiers	倍長整数	→	モディファイアーキーマスクまたはマスクの組み合わせ

説明

OBJECT SET SHORTCUTコマンドはobject と* で指定したオブジェクトのキーボードショートカットをカレントプロセスで動的に変更します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

key 引数にはオブジェクトに関連付けるキーを指定する文字列を渡します。以下のいずれかを渡すことができます:

- "B"のような標準のキーの名前
- Shortcut and Associated Keysテーマの定数やその値:

定数	型	値
Shortcut with Backspace	文字列	[backspace]
Shortcut with Carriage Return	文字列	[return]
Shortcut with Delete	文字列	[del]
Shortcut with Down Arrow	文字列	[down arrow]
Shortcut with End	文字列	[end]
Shortcut with Enter	文字列	[enter]
Shortcut with Escape	文字列	[esc]
Shortcut with F1	文字列	[F1]
Shortcut with F10	文字列	[F10]
Shortcut with F11	文字列	[F11]
Shortcut with F12	文字列	[F12]
Shortcut with F13	文字列	[F13]
Shortcut with F14	文字列	[F14]
Shortcut with F15	文字列	[F15]
Shortcut with F2	文字列	[F2]
Shortcut with F3	文字列	[F3]
Shortcut with F4	文字列	[F4]
Shortcut with F5	文字列	[F5]
Shortcut with F6	文字列	[F6]
Shortcut with F7	文字列	[F7]
Shortcut with F8	文字列	[F8]
Shortcut with F9	文字列	[F9]
Shortcut with Help	文字列	[help]
Shortcut with Home	文字列	[home]
Shortcut with Left Arrow	文字列	[left arrow]
Shortcut with Page Down	文字列	[page down]
Shortcut with Page Up	文字列	[page up]
Shortcut with Right Arrow	文字列	[right arrow]
Shortcut with Tabulation	文字列	[tab]
Shortcut with Up Arrow	文字列	[up arrow]

modifiers 引数にはショートカットに割り当てるひとつ以上のモディファイアーキーを渡します。modifiers 引数を設定するには、Events (Modifiers)テーマのひとつ以上の"mask"タイプ定数を渡します:

定数	型	値	コメント
Command key mask	倍長整数	256	Windows = Ctrlキー、Mac OS = Commandキー
Control key mask	倍長整数	4096	Mac OSのみ
Option key mask	倍長整数	2048	Windows = Altキー、Mac OS = Optionキー
Shift key mask	倍長整数	512	WindowsおよびMac OS

注: modifiers 引数を省略した場合、設定されたキーが押されるとそのオブジェクトが即座に有効になります。例えばボタンに"H"キーを関連付けた場合、Hキーを押すとボタンが押されたことになります。このような機能は特定のインターフェースに使用されます。

例題

アプリケーションの言語に基づき、異なるショートカットを割り当てたいとします。On Loadフォームイベントで以下のコードを実行します:

```
Case of
   :(vLang="JA")
      OBJECT SET SHORTCUT(*;"SortButton";"T";Command key mask+Shift key mask) // 日本語の場合Ctrl+Shift+T
   :(vLang="US")
      OBJECT SET SHORTCUT(*;"SortButton";"O";Command key mask+Shift key mask) // 英語の場合Ctrl+Shift+O
End case
```

OBJECT SET STYLE SHEET

OBJECT SET STYLE SHEET ({* ;} object ; styleSheetName)

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列)、省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
styleSheetName	テキスト	⇒	スタイルシート名

説明

OBJECT SET STYLE SHEETコマンドは、引数 objectと * で指定したオブジェクトと関連付けられたスタイルシートを、カレントのプロセスにおいて変更します。スタイルシートを変更すると、フォント、フォントサイズ、そしてフォントスタイルが変更されます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

styleSheetName 引数の中には、object 引数で指定したオブジェクトに適用するスタイルシートの名前等を渡します。渡せる値は以下の通りです。

- 既存のスタイルシート名(指定したスタイルシートが存在しない場合は、ON ERR CALL コマンドで割り込み可能なエラーが返されます)
- "Font Styles" テーマ内にある、Automatic style sheet 定数（指定すると、"自動"スタイルシートが適用されます）
- 空の文字列 ("")（指定すると、適用していたスタイルシートが解除されます。）

object 引数で指定したオブジェクトにデザインモードですでにスタイルシートが関連付けされている場合、このコマンドを呼び出すことによって、カレントプロセス中はスタイルシートが変更されます。

セッション中、ST SET ATTRIBUTES コマンド、ST SET TEXT コマンド、OBJECT SET FONT コマンド等をオブジェクトに使用してフォントやフォントサイズを変更した場合、スタイルシートへの参照はオブジェクトから自動的に削除されます。元からあったスタイルシートと同じ設定を適用しようとした場合でも同様に削除されます。
しかしながら、例えば ST SET ATTRIBUTES コマンドや OBJECT SET FONT STYLE コマンド等を使用してスタイル(太字、イタリック等)のみを変更した場合、スタイルシートへの参照は削除されず、これらのプロパティはセッションの間スタイルシートへと追加されます。

OBJECT SET SUBFORM

OBJECT SET SUBFORM ({* ;} object {; aTable}; detailSubform {; listSubform})

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
aTable	テーブル	→	フォームのテーブル (テーブルフォームの場合)
detailSubform	テキスト	→	サブフォームの詳細フォーム名
listSubform	テキスト	→	サブフォームのリストフォーム名 (テーブルフォーム)

説明

OBJECT SET SUBFORMコマンドはobjectと* 引数で指定したサブフォームオブジェクトに割り当てられる詳細フォームおよびオプションでリストフォームを動的に変更します。

注: このコマンドを使用してサブフォームのタイプ (プロパティリストの出力サブフォームプロパティ) を変更することはできません。このプロパティはデザインモードで設定しなければなりません。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

aTable 引数には使用するフォームが属するテーブルを渡します。ページモードのサブフォームとしてプロジェクトフォームを指定する場合、この引数を省略します。

detailSubform 引数には詳細サブフォームとして使用するフォームの名前を渡します。

listSubform 引数にはリストサブフォームとして使用するフォームの名前を渡します。この引数はリストタイプのサブフォームを変更する場合のみ指定できます。

ページモードのサブフォームを変更する場合、このコマンドをいつでも実行できます。カレントセレクションは変更されません。しかしリストサブフォームを変更する場合、リストを表示しているときのみ変更可能です。リスト中がダブルクリックされ詳細フォームが表示されている状態でこのコマンドを実行すると、エラーが生成されます。

OBJECT SET TEXT ORIENTATION

OBJECT SET TEXT ORIENTATION ({* ;} object ; orientation)

引数	型		説明
*	演算子	⇒	指定時: object はオブジェクト名(文字列) 省略時: object は変数またはフィールド
object	フォームオブジェクト	⇒	オブジェクト名(* 指定時)、または変数やフィールド(* 省略時)
orientation	倍長整数	⇒	オブジェクトの方向を決める値

説明

OBJECT SET TEXT ORIENTATIONコマンドは、引数 object と * で指定したオブジェクトの方向を、カレントプロセスにおいて変更します。

フォームエディター内の"方向"のプロパティは、フォームに対してテキストエリアの恒常的な回転を行います。このプロパティとは異なり、**OBJECT SET TEXT ORIENTATION** コマンドはオブジェクトそのものではなく、オブジェクトのコンテンツに対して回転を行います。詳細な情報に関しては *Design Reference* マニュアルを参照して下さい。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。
スタティックなテキストと、入力不可能な変数とフィールドのみ回転させることができます。このプロパティをサポートしないオブジェクトに対してコマンドを使用した場合、コマンドは何もしません。

orientation 引数には、オブジェクトに適用したい方向の絶対値を渡します。"Form Objects (Properties)" テーマ内にある、以下の定数のどれか一つのみ使用可能です:

定数	型	値	コメント
Orientation 0°	倍長整数	0	回転なし(デフォルト値)
Orientation 180°	倍長整数	180	テキストの方向を時計回りに180°回転
Orientation 90° left	倍長整数	270	テキストの方向を反時計回りに90°回転
Orientation 90° right	倍長整数	90	テキストの方向を時計回りに90°回転

注: 上記の値に対応する方向のみサポートされています。これ以外の値を渡した場合、その値は無視されます。

例題

フォームの内の変数に、270° の回転をさせたい場合:

```
OBJECT SET ENTERABLE(*;"myVar";False)
  // 変数が入力可能である場合は必須のコマンド
OBJECT SET TEXT ORIENTATION(*;"myVar";Orientation 90° left)
```

OBJECT SET THREE STATES CHECKBOX

OBJECT SET THREE STATES CHECKBOX ({* ;} object ; threeStates)

引数	型		説明
*	演算子	➡	指定時:object はオブジェクト名(文字列)、省略時:object は変数またはフィールド
object	フォームオブジェクト	➡	オブジェクト名(*指定時)、または変数やフィールド(*省略時)
threeStates	ブール	➡	True = スリーステートチェックボックス、False = 標準のチェックボックス

説明

OBJECT SET THREE STATES CHECKBOXコマンドは、引数 objectと * で指定したチェックボックスの"スリーステート"のプロパティを、カレントのプロセスにおいて変更します。

"スリーステート"オプションは、チェックボックスにて"どちらでもない"という新しい状態の使用を可能にします。詳細な情報に関しては*Design Reference* 内の 3 ステートチェックボックス を参照して下さい。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数でフィールドまたは変数を指定します。

このコマンドは変数と関連付けられたチェックボックスに対してのみ適用され、チェックボックスとして現れるブール型のフィールドには使用できません。

threeStates 引数には、スリーステートモードをオンにしたい場合には **True** を、オフにしたい場合は **False** を渡します。

OBJECT SET TITLE

OBJECT SET TITLE (* ; object ; title)

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
title	文字	→	オブジェクトの新しいタイトル

説明

OBJECT SET TITLEコマンドは、object 引数で指定されたボタンのタイトルをtitleで渡した値に変更します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。オブジェクト名に関する詳細はオブジェクトプロパティを参照してください。

OBJECT SET TITLEはタイトルを表示するオブジェクトに適用できます:

- ボタンと3Dボタン
- チェックボックスと3Dチェックボックス
- ラジオボタンと3Dラジオボタン
- リストボックスヘッダー
- スタティックテキストエリア
- グループボックス

通常このコマンドは一度に1つのオブジェクトに適用します。オブジェクトタイトルエリアにはテキストを表示するだけの十分な大きさが必要です。エリアが小さすぎると、テキストは途中までしか表示されません。titleにキャリッジリターンは使用しないでください。

複数行に渡るタイトルを設定したい場合、"\"文字(コードエディターでは"\\")を改行として使用して下さい。これは3Dボタン、3Dチェックすボックス、3Dラジオボタンにとリストボックスヘッダーに対して許可されています。

注: "\" をタイトルに使用したい際には "\\\" を渡して下さい。

例題 1

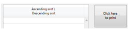
以下の例は、MODIFY SELECTIONを使用して表示された出力フォームのフッタエリアにある検索ボタンのオブジェクトメソッドです。このメソッドはテーブルを検索し、その検索結果に応じてbDeleteボタンを使用可または不可にして、そのボタンタイトルを変更します:

```
QUERY([People];[People]Name=vName)
Case of
   :(Records in selection([People])=0) // peopleが見つからなかった
      OBJECT SET TITLE(bDelete;" Delete")
      OBJECT SET ENABLED(bDelete;False)
   :(Records in selection([People])=1) // 一人見つかった
      OBJECT SET TITLE(bDelete;"Delete Person")
      OBJECT SET ENABLED(bDelete;True)
   :(Records in selection([People])>1) // 複数人見つかった
      OBJECT SET TITLE(bDelete;"Delete People")
      OBJECT SET ENABLED(bDelete;True)
End case
```

例題 2

タイトルを2行に渡って挿入したい場合:

```
OBJECT SET TITLE(*;"header1";"Ascending sort \\\ \\Descending sort")
OBJECT SET TITLE(*;"button1";"Click here \\to print")
```



OBJECT SET VERTICAL ALIGNMENT

OBJECT SET VERTICAL ALIGNMENT ({* ;} object ; alignment)

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
alignment	倍長整数	→	行揃えコード

説明

OBJECT SET VERTICAL ALIGNMENTコマンドはobject と* 引数で指定されたオブジェクトに適用される行揃えのタイプを変更します。

オプションの * 引数を渡すと、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合、object は変数であり、文字列ではなく変数参照を渡します。

alignment にはForm Objects (Properties)テーマの以下のいずれかの定数を渡します:

定数	型	値
Align bottom	倍長整数	4
Align center	倍長整数	3
Align default	倍長整数	1
Align top	倍長整数	2

行テキスト揃えは以下のオブジェクトに適用できます:

- リストボックス
- リストボックス列
- リストボックスヘッダーやフッター

OBJECT SET VISIBLE

OBJECT SET VISIBLE ({* ;} object ; visible)

引数	型		説明
*	演算子	➡	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	➡	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)
visible	ブール	➡	True: 表示, False: 非表示

説明

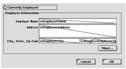
OBJECT SET VISIBLE コマンドは、objectによって指定されるオブジェクトを表示、あるいは非表示にします。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。オブジェクト名に関する詳細はを参照してください。

引数visibleにTrue渡すとオブジェクトが表示されます。引数visibleにFALSEを渡すとオブジェクトが非表示になります。

例題

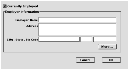
以下の図はデザインモードにおける典型的なフォームです:


Currently Employed
Employer Information
Employer Name
vsEmployerName
Address
vsEmployerAddress
City, State, Zip Code
vsEmployerCity
vsEmpl
vsEmployerZip
More...
Cancel
OK

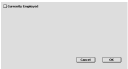
Employer Informationグループボックスにあるオブジェクトは、(グループボックスを含めて)どれもオブジェクトの名前に "employer"という文字が入っています。**Currently Employed**チェックボックスをオンにすると、オブジェクトが表示されます。チェックボックスをオフにすると、オブジェクトが表示されなくなります。
以下は、チェックボックスのオブジェクトメソッドです。

```
 // cbCurrentlyEmployed チェックボックスオブジェクトメソッド
Case of
   :(Form event=On Load)
      cbCurrentlyEmployed:=1

   :(Form event=On Clicked)
 // "emp"をオブジェクト名に含むオブジェクトを表示/非表示にする
      OBJECT SET VISIBLE(*;"@emp@";cbCurrentlyEmployed#0)
 ` チェックボックスは常に表示する
      OBJECT SET VISIBLE(cbCurrentlyEmployed;True)
End case
```

実行されるとフォームは以下のように表示されます:


☒ Currently Employed
Employer Information
Employer Name
Address
City, State, Zip Code
More...
Cancel
OK

または:


☐ Currently Employed
Cancel
OK

DISABLE BUTTON

DISABLE BUTTON ({* ;} object)

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)

互換性に関する注意

DISABLE BUTTON コマンドは4D v12より、廃止予定として定義されました。このコマンドは互換性の目的でのみ保持されます。このコマンドのスコープは指定された変数のすべてのインスタ ンスを含み、また関連とフォーム以外にも影響を及ぼします。これは"オブジェクトプロパティ"テーマのコマンドのそれに対応するものではありません。
ENABLE BUTTONとDISABLE BUTTONはそれぞれOBJECT SET ENABLEDとOBJECT Get enabledコマンドで置き換えることができます。

説明

DISABLE BUTTON コマンドは、objectで指定されたフォームブジェクトを使用不可にします。

使用不可のボタンやオブジェクトは、マウスクリックおよびショートカットに反応しません。

Note: ボタンやオブジェクトを使用不可にしても、その値をプログラムから変更することはできます。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。オブジェクト名に関する詳細はを参照してください。

このコマンドは、以下のようなオブジェクトタイプに適用されます:

- ボタン、デフォルトボタン、3Dボタン、透明ボタン、ハイライトボタン
- ラジオボタン、3Dラジオボタン、ラジオピクチャ
- チェックボックス、3Dチェックボックス
- ポップアップメニュー、ドロップダウンリストリスト、コンボボックス、メニュードロップダウンリスト
- サーモメータ、ルーラ

Note: (入力とキャンセルの動作を除く) 自動アクションを割り当てられたオブジェクトにこのコマンドを使用した場合、効果はありません。4Dが必要応じて状態を変更します。

例題 1

以下の例題はbValidateボタンを使用不可にします:

```
DISABLE BUTTON(bValidate)
```

例題 2

以下の例は、"btn"を名前に含むすべてのフォームオブジェクトを使用不可にします:

```
DISABLE BUTTON(*;"@btn@")
```

例題 3

OBJECT SET TITLEコマンドの例題参照

ENABLE BUTTON

ENABLE BUTTON ({* ;} object)

引数	型		説明
*	演算子	→	指定時, Objectはオブジェクト名 (文字列) 省略時, Objectはフィールドまたは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時), または フィールドまたは変数 (* 省略時)

互換性に関する注意

ENABLE BUTTON コマンドは4D v12より、廃止予定として定義されました。このコマンドは互換性の目的でのみ保持されます。このコマンドのスコープは指定された変数のすべてのインスタンスを含み、また関連とフォーム以外にも影響を及ぼします。これは"オブジェクトプロパティ"テーマのコマンドのそれに対応するものではありません。
ENABLE BUTTONとDISABLE BUTTONはそれぞれOBJECT SET ENABLEDとOBJECT Get enabledコマンドで置き換えることができます。

説明

ENABLE BUTTON コマンドは、objectで指定されたフォームオブジェクトを使用可能にします。

使用可能なボタンやオブジェクトは、マウスクリックおよびショートカットに反応します。

オプションの * 引数を指定した場合、objectはオブジェクト名です (文字列) 。オプションの * 引数を省略すると、objectはフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フィールドまたは変数のみ) を指定します。オブジェクト名に関する詳細はの節を参照してください。

このコマンドは、以下のようなオブジェクトタイプに適用されます:

- ボタン、デフォルトボタン、3Dボタン、透明ボタン、ハイライトボタン
- ラジオボタン、3Dラジオボタン、ラジオピクチャ
- チェックボックス、3Dチェックボックス
- ポップアップメニュー、ドロップダウンリストリスト、コンボボックス、メニュードロップダウンリスト
- サーモメータ、ルーラ

Note: (入力とキャンセルの動作を除く) 自動アクションを割り当てられたオブジェクトにこのコマンドを使用した場合、効果はありません。4Dが必要応じて状態を変更します。

例題 1

以下の例はbValidateボタンを使用可能にします:

```
ENABLE BUTTON(bValidate)
```

例題 2

以下の例は、"btn"を名前に含むすべてのフォームオブジェクトを使用可能にします:

```
ENABLE BUTTON(*;"@btn@")
```

例題 3

OBJECT SET TITLEコマンドの例題を参照

オブジェクト(ランゲージ)

4Dオブジェクトの構造

OB Copy New 14.0

OB Get New 14.0

OB GET ARRAY New 14.0

OB GET PROPERTY NAMES New 14.0

OB Get type New 14.0

OB Is defined New 14.0

OB Is empty New 14.0

OB REMOVE New 14.0

OB SET New 14.0

OB SET ARRAY New 14.0

OB SET NULL New 14.0

4Dオブジェクトの構造

オブジェクト(ランゲージ) テーマにはオブジェクトフォーム内のデータを作成、操作するためのコマンドが含まれています。この新機能によって、4Dは、構造化されたデータをサポートするどんなアプリケーションともデータのやりとりができるようになります。

4D オブジェクトは C_OBJECT ("コンパイラ" テーマ) または ARRAY OBJECT("配列" テーマ) コマンドを使用して作成・初期化することができます。

4D のネイティブなオブジェクトの構造は、よくある「値/名前」というペア(連想配列)に基づいています。これらのオブジェクトの記法は JSON のそれをもとにしていますが、完全に同じというわけではありません。

注: JSON オブジェクトを扱うためには新しい "JSON" テーマ内にあるコマンドを使用して下さい。

- プロパティ名は必ずテキストで表現されます。(例: "Name")
- プロパティ値は以下のどれかの型で表現されます:
    - 番号(実数、整数、等)
    - テキスト
    - 配列(テキスト配列、実数配列、倍長整数配列、整数配列、ブール配列、オブジェクト配列、ポインター配列)
    - null
    - ブール
    - ポインター(JSON Stringify コマンドを使用して評価されたか、コピーされたときにポインターとして保存されます)。
    - 日付("\"YYYY-MM-DDTHH:mm:ssZ\"" というフォーマットで表現されます)。
    - オブジェクト(オブジェクトを代入する階層の数に制限はありません)。

OB Copy

OB Copy (object {; resolvePtrs}) -> 戻り値

引数	型		説明
object	Object	⇒	構造化されたオブジェクト
resolvePtrs	ブール	⇒	True = ポインターを解決 False または省略時 = ポインターを解決しない
戻り値	Object	⇐	複製されたオブジェクト

説明

OB Copyコマンドは、objectのプロパティ、オブジェクト内オブジェクト、値を内包した完全なコピーのオブジェクトを返します。

object 引数で指定するオブジェクトは、C_OBJECT コマンドを使用して定義されている必要があります。

object で指定したオブジェクトがポインター型の値を格納している場合、複製先にもポインターが格納されます。もしくは、resolvePtrs 引数に **True** を渡すことで、複製時に値を解決させることもできます。この場合、オブジェクト内で値を指定しているポインターは解決され、解決済みの値が使用されます。

例題 1

単純な値を格納しているオブジェクトを複製する場合を考えます:

```
 C_OBJECT($Object)
 C_TEXT($JsonString)

 ARRAY OBJECT($arraySel;0)
 ALL RECORDS([Product])
 While(Not(End selection([Product])))
    OB SET($Object;"id";[Product]ID_Product)
    OB SET($Object;"Product Name";[Product]Product_Name)
    OB SET($Object;"Price";[Product]Price)
    OB SET($Object;"Tax rate";[Product]Tax_rate)
    $ref_value:=OB Copy($Object) //直接複製
    APPEND TO ARRAY($arraySel;$ref_value)
  //$ref_value の中身は $Object の中身と完全に同じ
    NEXT RECORD([Product])
 End while
  //$ref_value の中身
 $JsonString:=JSON Stringify array($arraySel)
```

例題 2

ポインターを格納しているオブジェクトを複製する場合:

```
 C_OBJECT($ref)

 OB SET($ref;"name";->[Company]name) //ポインターを含むオブジェクト
 OB SET($ref;"country";->[Company]country)
 ARRAY OBJECT($sel;0)
 ARRAY OBJECT($sel2;0)

 ALL RECORDS([Company])

 While(Not(End selection([Company])))
    $ref_comp:=OB Copy($ref) // copy without evaluating pointers
  // $ref_comp={"name":"->[Company]name","country":"->[Company]Country"}
    $ref_comp2:=OB Copy($ref;True) //解決済みのポインターを含んだコピー
  // $ref_comp={"name":"4D SAS","country":"France"}
    APPEND TO ARRAY($sel;$ref_comp)
    APPEND TO ARRAY($sel2;$ref_comp2)
    NEXT RECORD([Company])
 End while

 $Object:=JSON Stringify array($sel)
 $Object2:=JSON Stringify array($sel2)

  // $Object = [{"name":"","country":""},{"name":"","country":""},...]
  // $Object2 = [{"name":"4D SAS","country":"France"},{"name":"4D,
Inc","country":"USA"},{"name":"Catalan","country":"France"}...]
```

OB Get

OB Get (object ; property {; type}) -> 戻り値

引数	型		説明
object	Object	⇒	構造化されたオブジェクト
property	テキスト	⇒	情報を取得したいプロパティ名
type	倍長整数	⇒	値を変換したい型
戻り値	Object, テキスト, ブール, ポインター, 実数, 日付	⇐	プロパティのカレントの値

説明

OB Getコマンドは、object 引数で指定したオブジェクトの property のカレントの値を返します。任意の type 引数で指定した型へと変換することもできます。

object で指定するオブジェクトは、C_OBJECT コマンドを使用して作成されている必要があります。

property 引数には、情報を取得したいプロパティのラベルを渡します。property 引数では、大文字と小文字は区別されることに注意して下さい。

特に指定がなければ、4D はプロパティの値を本来の型のまま返します。このとき、type 引数を使用することによって返ってくる値の型を強制的に変換することができます。この場合、type 引数には以下の定数のどれか一つを渡して下さい。これらの定数は Field and Variable Types テーマ内にあります。

定数	型	値
Is Boolean	倍長整数	6
Is Date	倍長整数	4
Is Integer	倍長整数	8
Is Integer 64 bits	倍長整数	25
Is JSON null	倍長整数	255
Is LongInt	倍長整数	9
Is object	倍長整数	38
Is Real	倍長整数	1
Is String Var	倍長整数	24
Is Text	倍長整数	2
Is Time	倍長整数	11
Object array	倍長整数	39

コマンドは property 引数で指定されたプロパティの値を返します。いくつかのデータの型がサポートされています。以下のことに注意して下さい:

- ポインターはそのまま返されますが、JSON Stringify コマンドを使用することによってポインターが指してる値に変換することができます。
- 日付は "\"YYYY-MM-DDTHH:mm:ssZ\"" というフォーマットで返されます。
- 実数の値において、小数点はかならずピリオド(".")になります。
- 時刻は数字で表現されます(4Dでは秒単位の、JSONではミリ秒単位の数字になります)。

例題 1

テキスト型の値を取得する場合:

```
C_OBJECT($ref)
C_TEXT($FirstName)
OB SET($ref;"FirstName";"Harry")
$FirstName:=OB Get($ref;"FirstName") // $FirstName = "Harry" (text)
```

例題 2

実数型の値を取得して、倍長整数へと変換する場合:

```
OB SET($ref ;"age";42)
$age:=OB Get($ref ;"age") //  $age は実数型(デフォルト)
$age:=OB Get($ref ;"age";Is LongInt) // $age を倍長整数型に変換
```

例題 3

オブジェクトの中の値を取得する場合:

```
C_OBJECT($ref1;$ref2)
OB SET($ref1;"LastName";"Smith") // $ref1={"LastName":"Smith"}
OB SET($ref2;"son";$ref1) // $ref2={"son":{"LastName":"Smith"}}
$son:=OB Get($ref2;"son") // $son={"LastName":"john"} (object)
$sonsName:=OB Get($son ;"name") // $sonsName="john" (text)
```

例題 4

従業員の年齢を二度修正したい場合:

```
C_OBJECT($ref_john;$ref_jim)
OB SET($ref_john;"name";"John";"age";35)
OB SET($ref_jim;"name";"Jim";"age";40)
APPEND TO ARRAY($myArray;$ref_john) // オブジェクト配列を作成
APPEND TO ARRAY($myArray;$ref_jim)
 // John の年齢を35から25 へと修正
OB SET($myArray{1};"age";25)
 // 配列内で"John" の年齢を修正
For($i;1;Size of array($myArray))
   If(OB Get($myArray{$i};"name")="John")
      OB SET($myArray{$i};"age";36) // 25 から 36 へと修正
 // $ref_john={"name":"John","age":36}
   End if
End for
```

例題 5

ISO 形式のデータを日付型で取り出す場合:

```
C_OBJECT($object)
C_DATE($birthday)
C_TEXT($birthdayString)
OB SET($object;"Birthday";"1990-12-25T12:00:00Z")
$birthdayString:=OB Get($object;"Birthday")
 // $birthdayString="1990-12-25T12:00:00Z"
$birthday:=OB Get($object;"Birthday";Is Date)
 // $birthday=25/12/90
```

例題 6

入れ子にされたオブジェクトを使用することもできます:

```
C_OBJECT($ref1;$child;$children)
C_TEXT($childName)
OB SET($ref1;"firstname";"John";"lastname";"Monroe")
 //{"firstname":"john","lastname";"Monroe"}
OB SET($children;"children";$ref1)
$child:=OB Get($children;"children")
 //$son = {"firstname":"John","lastname":"Monroe"} (object)
$childName:=OB Get($child;"lastname")
 //$childName = "Monroe" (text)
 // または
$childName:=OB Get(OB Get($children;"children");"lastname")
 // $childName = "Monroe" (text)
```

OB GET ARRAY

OB GET ARRAY (object ; property ; array)

引数	型		説明
object	Object	⇒	構造化されたオブジェクト
property	テキスト	⇒	情報を取得したいプロパティ名
array	テキスト配列, 実数配列, ブール配列, Object array, ポインター配列	⇐	プロパティの値の配列

説明

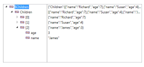
OB GET ARRAYコマンドは、object 引数で指定したランゲージオブジェクトの、property 引数で指定したプロパティの中に保存されている値の配列を array という配列に返します。

object で指定するオブジェクトは、C_OBJECT コマンドを使用して作成されている必要があります。

property 引数には、値を取得したいプロパティのラベルを渡します。property 引数では、大文字と小文字は区別されることに注意して下さい。

例題 1

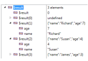
OB SET ARRAY コマンドの例示で定義された以下のオブジェクトにおいて:



```
◢ $Children        {"Children":[{"name":"Richard","age":7},{"name":"Susan","age":4}...
   ◢ Children      [{"name":"Richard","age":7},{"name":"Susan","age":4},{"name":"J...
      ▷ [0]        {"name":"Richard","age":7}
      ▷ [1]        {"name":"Susan","age":4}
      ◢ [2]        {"name":"James","age":3}
           age     3
           name    "James"
```

以下の値を取得したい場合は以下の様になります:

```
ARRAY OBJECT($result;0)
OB GET ARRAY($Children;"Children";$result)
```



```
◢ $result          3 elements
     $result       0
   ▷ $result{0}    undefined
   ◢ $result{1}    {"name":"Richard","age":7}
        age        7
        name       "Richard"
   ◢ $result{2}    {"name":"Susan","age":4}
        age        4
        name       "Susan"
   ▷ $result{3}    {"name":"James","age":3}
```

例題 2

配列の第一要素の値を変更したい場合、以下の様になります:

```
  // "age" の値を変更:
ARRAY OBJECT($refs)
OB GET ARRAY($refEmployees;"__ENTITIES";$refs)
OB SET($refs{1};"age";25)
```

OB GET PROPERTY NAMES

OB GET PROPERTY NAMES (object ; arrProperties {; arrTypes})

引数	型		説明
object	Object	→	構造化されたオブジェクト
arrProperties	テキスト配列	←	プロパティ名
arrTypes	倍長整数配列	←	プロパティの型

説明

OB GET PROPERTY NAMESコマンドは、object 引数で指定したランゲージオブジェクトの中に含まれているプロパティの名前を、arrProperties という名前のテキスト配列に入れて返します。

object で指定するオブジェクトは、C_OBJECT コマンドを使用して作成されている必要があります。

arrProperties 引数にはテキスト配列を渡します。配列が存在しない場合、コマンドが自動的に作成してリサイズします。

また、任意の arrTypes 引数に倍長整数配列を渡すこともできます。この場合、arrProperties 内の個々の要素に関して、プロパティに保存された値の型を arrTypes に返します。返される値は、"Field and Variable Types" テーマ内にある以下の定数のどれかになります:

定数	型	値
Is Boolean	倍長整数	6
Is JSON null	倍長整数	255
Is object	倍長整数	38
Is Real	倍長整数	1
Is String Var	倍長整数	24
Is Text	倍長整数	2
Is Undefined	倍長整数	5
Object array	倍長整数	39

例題 1

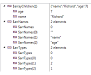
オブジェクトが空でないかどうかをテストしたい場合を考えます:

```
ARRAY TEXT(arrNames;0)
ARRAY LONGINT(arrTypes;0)
C_OBJECT($ref_richard)
OB SET($ref_richard;"name";"Richard";"age";7)
OB GET PROPERTY NAMES($ref_richard;arrNames;arrTypes)
 // arrNames{1}="name", arrNames{2}="age"
 // arrTypes{1}=2, arrTypes{2}=1
If(Size of array(arrNames)#0)
 // ...
End if
```

例題 2

オブジェクト配列の要素を使用する場合を考えます:

```
C_OBJECT($Children;$ref_richard;$ref_susan;$ref_james)
ARRAY OBJECT($arrayChildren;0)

OB SET($ref_richard;"name";"Richard";"age";7)
APPEND TO ARRAY($arrayChildren;$ref_richard)
OB SET($ref_susan;"name";"Susan";"age";4)
APPEND TO ARRAY($arrayChildren;$ref_susan)
OB SET($ref_james;"name";"James";"age";3)
APPEND TO ARRAY($arrayChildren;$ref_james)

OB GET PROPERTY NAMES($arrayChildren{1};$arrNames;$arrTypes)
```



$arrayChildren{1}	{"name":"Richard","age":7}
age	7
name	"Richard"
$arrNames	2 elements
$arrNames	0
$arrNames{0}	""
$arrNames{1}	"name"
$arrNames{2}	"age"
$arrTypes	2 elements
$arrTypes	0
$arrTypes{0}	0
$arrTypes{1}	2
$arrTypes{2}	1

OB Get type

OB Get type (object ; property) -> 戻り値

引数	型		説明
object	Object	➡	構造化されたオブジェクト
property	テキスト	➡	読み出したいプロパティ名
戻り値	倍長整数	↩	プロパティの値のタイプ

説明

OB Get typeコマンドは、object で指定したランゲージオブジェクトの property に関連付けられた値の型を返します。

object で指定するオブジェクトは、C_OBJECT コマンドを使用して作成されている必要があります。

property 引数には、値の型を取得したいプロパティのラベルを渡します。
property 引数では、大文字と小文字は区別されることに注意して下さい。

コマンドは、指定した値の型を示す倍長整数の値を返します。返される値は、"Field and Variable Types" テーマ内にある以下の定数のどれかになります。

定数	型	値
Is Boolean	倍長整数	6
Is JSON null	倍長整数	255
Is object	倍長整数	38
Is Real	倍長整数	1
Is String Var	倍長整数	24
Is Text	倍長整数	2
Is Undefined	倍長整数	5
Object array	倍長整数	39

例題

よくある値の型を取得する場合:

```
C_OBJECT($ref)
OB SET($ref;"name";"smith";"age";42)
$type:=OB Get type($ref;"name") //$type returns 2
$type2:=OB Get type($ref;"age") //$type2 returns 1
```

OB Is defined

OB Is defined (object {; property}) -> 戻り値

引数	型	説明
object	Object	→ 構造化されたオブジェクト
property	テキスト	→ 指定時にはプロパティをチェック、省略時にはオブジェクトをチェック
戻り値	ブール	↩ property 省略時:object が定義済みの場合はTrue を、それ以外は False を返す property 指定時:property が定義済みの場合はTrue を、それ以外は False を返す

説明

OB Is definedコマンドは、object または property が定義済みである場合にはTrueを返し、それ以外の場合にはFalseを返します。

object 引数で指定するオブジェクトは、C_OBJECT コマンドを使用して作成されたものでなければなりません。

property 引数を省略した場合、コマンドは object が定義済みかどうかを調べます。オブジェクトの中身が初期化されていれば定義済みであるとみなされます。

注: 定義済みのオブジェクトでも空である場合もあります。オブジェクトが空もしくは未定義であるかどうかを調べる場合には、OB Is empty コマンドを使用して下さい。

property 引数を渡した場合、コマンドは指定されたプロパティが object 内に存在するかどうかをチェックします。property 引数では、大文字と小文字は区別されることに注意して下さい。

例題 1

オブジェクトの初期化を調べる記法:

```
C_OBJECT($object)
$def:=OB Is defined($object) // $object は未定義なので$def=false

OB SET($object;"Name";"Martin")
OB REMOVE($object;"Name")
$def2:=OB Is defined($object) // $object は空({})ではあるものの定義済みなので$def2=true
```

例題 2

プロパティが存在しているかどうかを調べます:

```
C_OBJECT($ref)
OB SET($ref;"name";"smith";"age";42)
 //...
If(OB Is defined($ref;"age"))
 //...
End if
```

このテストは以下のコマンドと同等です:

```
If(OB Get type($Object;"name")#Is Undefined)
```

OB Is empty

OB Is empty (object) -> 戻り値

引数	型		説明
object	Object	→	構造化されたオブジェクト
戻り値	ブール	↩	object が空か未定義のときには True 、それ以外のときには False

説明

OB Is emptyコマンドは、object が未定義か空である場合にはTrueを返し、object が定義済み(初期化済み)で少なくとも一つのプロパティを内包している場合にはFalseを返します。

object で指定するオブジェクトは、C_OBJECT コマンドを使用して作成されている必要があります。

例題

OB Is defined コマンド同様、このコマンドもオブジェクトの中身によって異なる結果を返します:

```
C_OBJECT($ref)
$empty:=OB Is empty($ref) // True
$def:=OB Is defined($ref) // False

OB SET($ref;"name";"Susie";"age";4)
 // $ref="{"name":"Susie","age":4}"
$empty:=OB Is empty($ref) // False
$def:=OB Is defined($ref) // True

OB REMOVE($ref;"name")
OB REMOVE($ref;"age")
$empty:=OB Is empty($ref) // True
$def:=OB Is defined($ref) // True
```

OB REMOVE

OB REMOVE (object ; property)

引数	型		説明
object	Object	⇒	構造化されたオブジェクト
property	テキスト	⇒	削除したいプロパティの名前

説明

OB REMOVEコマンドは、object 引数で指定したランゲージオブジェクトの、property 引数で指定したプロパティを削除します。

object で指定するオブジェクトは、C_OBJECT コマンドを使用して作成されている必要があります。

property 引数には、削除したいプロパティのラベルを渡します。
property 引数では、大文字と小文字は区別されることに注意して下さい。

例題

オブジェクトから、"age" のプロパティを削除する場合:

```
C_OBJECT($Object)
OB SET($Object;"name";"smith";"age";42;"client";True)
 // $Object={"name":"smith","age":42,"client":true}
OB REMOVE($Object;"age")
 // $Object={"name":"smith","client":true}
```

OB SET

OB SET (object ; property ; value {; property2 ; value2 ; ... ; propertyN ; valueN})

引数	型		説明
object	Object	→	構造化されたオブジェクト
property	テキスト	→	設定したいプロパティの名前
value	テキスト, 日付, ブール, ポインター, Number, Object	→	プロパティの新しい値

説明

OB SETコマンドは、object 引数で指定したランゲージオブジェクトの中に、一つ以上の プロパティ/値 のペアを作成もしくは変更します。

object で指定するオブジェクトは、C_OBJECT コマンドを使用して作成されている必要があります。

property 引数には、作成または修正したいプロパティのラベル(名前)を渡して下さい。object 内に指定されたプロパティが存在する場合、その値は指定した値で上書きされます。プロパティが存在しない場合、新たにプロパティが作成されます。
property 引数では、大文字と小文字は区別されることに注意して下さい。

value 引数には、プロパティに設定したい値を渡して下さい。渡せる値としては複数の型がサポートされています。渡す際には以下のことに注意して下さい:

- ポインターを渡した場合、それはそのままの形で保存されます。値を解決するためには JSON Stringify コマンドを使用します。
- 日付を渡す場合には、"\"YYYY-MM-DDTHH:mm:ssZ\"" というフォーマットで表現されている必要があります。
- ランゲージオブジェクトを渡した場合、コマンドはオブジェクトの参照を使用し、実際にコピーを作成するわけではありません。

例題 1

オブジェクトを作成し、テキスト型のプロパティを追加する場合を考えます:

```
C_OBJECT($Object)
OB SET($Object ;"FirstName";"John";"LastName";"Smith")
 // $Object = {"FirstName":"John","LastName":"Smith"}
```

例題 2

オブジェクトを作成し、ブール型のプロパティを追加する場合を考えます:

```
C_OBJECT($Object)
OB SET($Object ;"LastName";"smith";"age";42;"client";True)
 // $Object = {"LastName":"smith","age":42,"client":true}
```

例題 3

プロパティを修正する場合:

```
 // $Object = {"FirstName":"John","LastName":"Smith"}
OB SET($Object ;"FirstName";"Paul")
 // $Object = {"FirstName":"Paul","LastName":"Smith"}
```

例題 4

プロパティを追加する場合:

```
 // $Object = {"FirstName":"John","LastName":"Smith"}
OB SET($Object ;"department";"Accounting")
 // $Object = {"FirstName":"Paul","LastName":"Smith","department":"Accounting"}
```

例題 5

プロパティの名前を変更する場合:

```
C_OBJECT($Object)
OB SET($Object ;"LastName";"James";"age";35)
 // $Object = {"LastName":"James","age":35}
OB SET($Object ;"FirstName";OB Get($Object ;"LastName"))
 // $Object = {"FirstName":""James","nom":"James","age":35}
OB REMOVE($Object ;"LastName")
 // $Object = {"FirstName":""James","age":35}
```

例題 6

ポインターを使用する場合:

```
 // $Object = {"FirstName":"Paul","LastName":"Smith"}
C_TEXT($LastName)
OB SET($Object ;"LastName";->$LastName)
 // $Object = {"FirstName":"Paul","LastName":"->$LastName"}
$JsonString:=JSON Stringify($Object)
 // $JsonString="{"FirstName":"Paul","LastName":""}
$LastName:="Wesson"
$JsonString:=JSON Stringify($Object)
 // $JsonString="{"FirstName":"Paul","LastName":"Wesson"}
```

例題 7

オブジェクトを使用する場合:

```
C_OBJECT($ref_smith)
OB SET($ref_smith ;"name";"Smith")
C_OBJECT($ref_emp)
OB SET($ref_emp ;"employee";$ref_smith)
$Json_string :=JSON Stringify($ref_emp)
 // $ref_emp = {"employee":{"name":"Smith"}} (object)
 // $Json_string = "{"employee":{"name":"Smith"}}" (string)
```

値をプログラム実行中に変えることもできます:

```
OB SET($ref_smith ;"name";"Smyth")
 // $ref_smith = {"employee":{"name":"Smyth"}}
$string:=JSON Stringify($ref_emp)
 // $string = "{"employee":{"name":"Smyth"}}"
```

例題 8

オブジェクト配列を使用する場合:

```
C_TEXT($jsonString)
C_OBJECT($Contact)
OB SET($Contact;"FirstName";"Alan")
OB SET($Contact;"LastName";"Monroe")
OB SET($Contact;"age";40)
OB SET($Contact;"telephone";"[2128675309,2128671234]")

$jsonString:=JSON Stringify($Contact)

 // $Contact の値:
 // {"LastName":"Monroe","telephone":"[2128675309,2128671234]",
 // "age":40,"FirstName":"Alan"}

 // $jsonString の値:
 // "{"LastName":"Monroe","telephone":"[2128675309,2128671234]",
 // "age":40,"FirstName":"Alan"}"
```

OB SET ARRAY

OB SET ARRAY (object ; property ; array)

引数	型		説明
object	Object	➡	構造化されたオブジェクト
property	テキスト	➡	設定したいプロパティ名
array	テキスト配列, 実数配列, ブール配列, Object array, ポインター配列, 倍長整数配列	➡	プロパティに保存したい配列

説明

OB SET ARRAYコマンドは、object 引数で指定したランゲージオブジェクトの中の propertyと関連付ける配列 array を定義します。

object で指定するオブジェクトは、C_OBJECT コマンドを使用して作成されている必要があります。

property 引数には、作成または修正したいプロパティのラベル(名前)を渡して下さい。オブジェクト内に指定されたプロパティが存在する場合、その値は指定した値で上書きされます。プロパティが存在しない場合、新たにプロパティが作成されます。
property 引数では、大文字と小文字は区別されることに注意して下さい。

array 引数には、プロパティの値として関連付けたい配列を渡して下さい。渡せる配列の型としては複数の型がサポートされています。

注: 二次元配列を使用することはできません。

例題 1

テキスト配列を使用する場合:

```
 C_OBJECT($Children)
 ARRAY TEXT($arrChildren;3)
 $arrChildren{1}:="Richard"
 $arrChildren{2}:="Susan"
 $arrChildren{3}:="James"

 OB SET ARRAY($Children;"Children";$arrChildren)
  // Value of $Children = {"Children":["Richard","Susan","James"]}
```

例題 2

配列に要素を追加する場合:

```
 ARRAY TEXT($arrText;2)
 $arrText{1}:="Smith"
 $arrText{2}:="White"
 C_OBJECT($Employees)
 OB SET ARRAY($Employees;"Employees";$arrText)
 APPEND TO ARRAY($arrText;"Brown") // Add to the 4D array
  // $Employees = {"Employees":["Smith","White"]}

 OB SET ARRAY($Employees;"Employees";$arrText)
  // $Employees = {"Employees":["Smith","White","Brown"]}
```

例題 3

テキスト配列の一要素を配列として使用する場合:

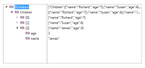
```
  // $Employees = {"Employees":["Smith","White","Brown"]}
 OB SET ARRAY($Employees ;"Manager";$arrText{1})
  // $Employees = {"Employees":["Smith","White","Brown"],"Manager":["Smith"]}
```

例題 4

オブジェクト配列を使用する場合:

```
 C_OBJECT($Children;$ref_richard;$ref_susan;$ref_james)
 ARRAY OBJECT($arrChildren;0)
 OB SET($ref_richard;"nom";"Richard";"age";7)
 APPEND TO ARRAY($arrChildren;$ref_richard)
 OB SET($ref_susan;"name";"Susan";"age";4)
 APPEND TO ARRAY($arrChildren;$ref_susan)
 OB SET($ref_james;"name";"James";"age";3)

 APPEND TO ARRAY($arrChildren;$ref_james)

  // $arrChildren {1} = {"name":"Richard","age":7}
  // $arrChildren {2} = {"name":"Susan","age":4}
  // $arrChildren {3} = {"name":"James","age":3}

 OB SET ARRAY($Children;"Children";$arrChildren)

  // $Children = {"Children":[{"name":"Richard","age":7},{"name":"Susan",
  // "age":4},{"name":"James","age":3}]}
```

オブジェクトはデバッガ内では以下の様に表示されます:



```
◢ $Children        {"Children":[{"name":"Richard","age":7},{"name":"Susan","age":4}...
  ◢ Children       [{"name":"Richard","age":7},{"name":"Susan","age":4},{"name":"J...
    ▷ [0]          {"name":"Richard","age":7}
    ▷ [1]          {"name":"Susan","age":4}
    ◢ [2]          {"name":"James","age":3}
        age        3
        name       "James"
```

OB SET NULL

OB SET NULL (object ; property)

引数	型	説明
object	Object	⇒ 構造化されたオブジェクト
property	テキスト	⇒ null 値を適用したいプロパティ名

説明

OB SET NULLコマンドは、object 引数で指定したランゲージオブジェクトの中に**null**値を保存します。

object で指定するオブジェクトは、C_OBJECT コマンドを使用して作成されている必要があります。

property 引数には、**null** の値を保存したいプロパティのラベル(名前)を指定します。オブジェクト内に指定されたプロパティが存在する場合、その値は **null** で上書きされます。プロパティが存在しない場合、新たにプロパティが作成されます。
property 引数では、大文字と小文字は区別されることに注意して下さい。

例題

Lea の "age" というプロパティにnull を入れる場合:

```
C_OBJECT($ref)
OB SET($ref;"name";"Lea";"age";4)
 // $ref = {"name":"Lea","age":4)
...
OB SET NULL($ref ;"age")
 // $ref = {"name":"Lea","age":null}
```

クイックレポート

- QR BLOB TO REPORT
- QR Count columns
- QR DELETE COLUMN
- QR DELETE OFFSCREEN AREA
- QR EXECUTE COMMAND
- QR Find column
- QR Get area property
- QR GET BORDERS
- QR Get command status
- QR GET DESTINATION
- QR Get document property
- QR Get drop column
- QR GET HEADER AND FOOTER
- QR Get HTML template
- QR GET INFO COLUMN
- QR Get info row
- QR Get report kind
- QR Get report table
- QR GET SELECTION
- QR GET SORTS
- QR Get text property
- QR GET TOTALS DATA
- QR GET TOTALS SPACING
- QR INSERT COLUMN
- QR New offscreen area
- QR ON COMMAND Updated 14.0
- QR REPORT Updated 14.0
- QR REPORT TO BLOB
- QR RUN
- QR SET AREA PROPERTY
- QR SET BORDERS
- QR SET DESTINATION Updated 14.0
- QR SET DOCUMENT PROPERTY
- QR SET HEADER AND FOOTER
- QR SET HTML TEMPLATE
- QR SET INFO COLUMN
- QR SET INFO ROW
- QR SET REPORT KIND
- QR SET REPORT TABLE
- QR SET SELECTION
- QR SET SORTS
- QR SET TEXT PROPERTY
- QR SET TOTALS DATA
- QR SET TOTALS SPACING

QR BLOB TO REPORT

QR BLOB TO REPORT (area ; BLOB)

引数	型		説明
area	倍長整数	→	エリア参照
BLOB	BLOB	→	レポートを納めたBLOB

説明

QR BLOB TO REPORT コマンドは、blobに格納されたレポートをareaに渡されたクイックレポートエリアに配置します。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なblob引数を渡した場合、エラー番号-9852が生成されます。

例題 1

次のコードを使用し、データベースストラクチャと同じ階層にある"report.4qr"という名称のレポートファイルをMyAreaに表示することができます。このレポートファイルは4D 2003で作成されている必要はなく、以前のバージョンのものであっても構いません:

```
C_BLOB($doc)
C_LONGINT(MyArea)
DOCUMENT TO BLOB("report.4qr";$doc)
QR BLOB TO REPORT(MyArea;$doc)
```

例題 2

次の例は、Field4に格納されたクイックレポートを取り出し、MyAreaに表示します:

```
QR BLOB TO REPORT(MyArea;[Table 1]Field4)
```

QR Count columns

QR Count columns (area) -> 戻り値

引数	型		説明
area	倍長整数	→	エリア参照
戻り値	倍長整数	↩	エリア中のカラム数

説明

QR Count columns コマンドは、クイックレポートareaに存在するカラムの数を返します。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

例題

次のコードはカラムの数を取得して、一番右端にある既存のカラムの右側にカラムを挿入します:

```
$ColNb:=QR Count columns(MyArea)
QR INSERT COLUMN(MyArea;$ColNb+1;->[Table1]Field2)
```

QR DELETE COLUMN

QR DELETE COLUMN (area ; colNumber)

引数	型		説明
area	倍長整数	⇒	エリア参照
colNumber	倍長整数	⇒	カラム番号

説明

QR DELETE COLUMNは、areaにあるcolNumberに渡された番号のカラムを削除します。このコマンドはクロステーブルレポートに対しては適用されません。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なcolumn番号を渡した場合、エラー番号-9852が生成されます。

例題

次の例題は、レポートがリストレポートであることを確認し、3番目のカラムを削除します:

```
If(QR Get report kind(MyArea)=qr list report)
   QR DELETE COLUMN(MyArea;3)
End if
```

QR DELETE OFFSCREEN AREA

QR DELETE OFFSCREEN AREA (area)

引数	型		説明
area	倍長整数	→	削除するエリアの参照

説明

QR DELETE OFFSCREEN AREA コマンドは、area引数に渡された参照番号のクイックレポートオフスクリーンエリアをメモリから削除します。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR EXECUTE COMMAND

QR EXECUTE COMMAND (area ; command)

引数	型		説明
area	倍長整数	⇒	エリア参照
command	倍長整数	⇒	実行するメニューコマンド

説明

QR EXECUTE COMMAND コマンドは、commandに渡された参照番号のメニューコマンドまたはツールバーボタンを実行します。このコマンドの最も一般的な使い方は、ユーザが選択したコマンドをQR ON COMMANDで中断し、その後にコマンドを実行することです。

commandには、値またはテーマの定数を渡します。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なcommand引数を渡した場合、エラー番号-9852が生成されます。

QR Find column

QR Find column (area ; expression) -> 戻り値

引数	型		説明
area	倍長整数	→	エリア参照
expression	文字, ポインター	→	カラムオブジェクト
戻り値	倍長整数	↩	カラム番号

説明

QR Find column コマンドは、expression引数で渡された表現式に一致する内容を持つ最初のカラムの番号を返します。

expression には文字列またはポインタを渡します。

対象となるカラムが見つからない場合、は-1を返します。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

例題

次のコードは、[G.NQR Tests]Quarterフィールドが格納されているカラム番号を取得し、そのカラムを削除します:

```
$NumColumn:=QR Find column(MyArea;->[G.NQR Tests]Quarter)
```

または:

```
$NumColumn:=QR Find column(MyArea;"[G.NQR Tests]Quarter")
```

そして:

```
If($NumColumn#-1)
   QR DELETE COLUMN(MyArea;$NumColumn)
End if
```

QR Get area property

QR Get area property (area ; property) -> 戻り値

引数	型		説明
area	倍長整数	→	エリアの参照
property	倍長整数	→	インタフェース要素
戻り値	倍長整数	↩	1 = 表示, 0 = 非表示

説明

QR Get area property コマンドは、propertyに渡されたインタフェース要素（ツールバーまたはメニューバー）が表示されていなければ0を、表示されていれば1を返します。

メニューバーとツールバーには1から6まで (上から下へ) の番号が付けられ、値7はコンテキストメニュー専用です。

テーマの定数を使用し、インタフェース要素を指定することができます:

定数	型	値	コメント
qr view color toolbar	倍長整数	5	カラーツールバーのステータス取得 (表示=1, 非表示=0)
qr view column toolbar	倍長整数	6	カラムツールバーのステータス取得 (表示=1, 非表示=0)
qr view contextual menus	倍長整数	7	コンテキストメニューのステータス取得 (表示=1, 非表示=0)
qr view menubar	倍長整数	1	メニューバーのステータス取得 (表示=1, 非表示=0)
qr view operators toolbar	倍長整数	4	関数ツールバーのステータス取得 (表示=1, 非表示=0)
qr view standard toolbar	倍長整数	2	標準ツールバーのステータス取得 (表示=1, 非表示=0)
qr view style toolbar	倍長整数	3	スタイルツールバーのステータス取得 (表示=1, 非表示=0)

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なproperty引数を渡した場合、エラー番号-9852が生成されます。

QR GET BORDERS

QR GET BORDERS (area ; column ; row ; border ; row | level {; color})

引数	型		説明
area	倍長整数	→	エリア参照
column	倍長整数	→	カラム番号
row	倍長整数	→	行番号
border	倍長整数	←	
row | level	倍長整数	←	線の太さ
color	倍長整数	←	罫線のカラー

説明

QR GET BORDERS コマンドを使用し、指定したセルの罫線スタイルを取得できます。

areaにはクイックレポートエリアの参照番号を渡します。

columnにはセルのカラム番号を渡します。

rowにはセルの行番号を渡します:

- 対象の小計 (ブレーク) レベルを指定する正の整数を渡す、または
- テーマの以下の定数のいずれかを渡す

定数	型	値	コメント
qr detail	倍長整数	-2	レポートの詳細エリア
qr grand total	倍長整数	-3	総計エリア
qr title	倍長整数	-1	レポートタイトル

borderには、適用するセルの罫線を示す値を渡します。テーマのいずれかの値を渡します:

定数	型	値	コメント
qr bottom border	倍長整数	8	下罫線
qr inside horizontal border	倍長整数	32	内側縦罫線
qr inside vertical border	倍長整数	16	内側横罫線
qr left border	倍長整数	1	左罫線
qr right border	倍長整数	4	右罫線
qr top border	倍長整数	2	上罫線

Note: QR SET BORDERSコマンドとは異なり、QR GET BORDERSは累計値を受け付けません。罫線の全てのパラメータを知るには、全ての罫線値を使って個別にテストする必要があります。

lineにはその線の太さを返します:

- 0 = 線なし
- 1 = 1/4ポイント
- 2 = 1/2ポイント
- 3 = 1ポイント
- 4 = 2ポイント

colorはその線の色を返します。返される値は、指定した罫線部分のカラー値です。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なcolumn引数を渡した場合、エラー番号-9852が生成されます。
無効なrow引数を渡した場合、エラー番号-9853が生成されます。
無効なborder引数を渡した場合、エラー番号-9854が生成されます。

QR Get command status

QR Get command status (area ; command {; value}) -> 戻り値

引数	型	説明
area	倍長整数	⇒ エリア参照
command	倍長整数	⇒ コマンド番号
value	倍長整数, テキスト	⇐ 選択されたサブアイテムの値
戻り値	倍長整数	↩ コマンドの状態

説明

QR Get command status コマンドは、commandが使用不可であれば0を、使用可能であれば1を返します。

valueは、選択されたサブアイテムがあれば、その値を返します。例えば、選択されたコマンドが**フォント**メニュー (1000) であり、選択されたフォントが"Arial"である場合、valueには"Arial"が返されます。また、選択されたコマンドが**カラー**メニュー (1002,、1003、または1004) である場合、valueにはカラー番号が返されます。

このコマンドは、次の2つの状況で使用することができます:

- あるコマンドが使用可であるか使用不可であるかを調べる単純な判定文として。
- QR ON COMMANDコマンドでインストールされたメソッドにおいてこのコマンドを使用すると、選択されたサブアイテムを知ることができます。そのメソッドでは、$1がエリアの参照番号となり、$2がコマンド番号となります。

commandにはには値または定数テーマの定数を渡すことができます。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なcommand引数を渡した場合、エラー番号-9852が生成されます。

QR GET DESTINATION

QR GET DESTINATION (area ; type {; specifics})

引数	型		説明
area	倍長整数	→	エリア参照
type	倍長整数	←	レポートのタイプ
specifics	文字, 変数	←	出力先の詳細

説明

QR GET DESTINATION コマンドは、areaに渡したエリア参照のレポート出力先を取得します。

引数typeの値をテーマの定数と比較することができます。

次の表はtypeおよびspecificsの両引数から取得できる値を示しています:

出力先	タイプ (値)	詳細
プリンタ	qr printer (1)	N.A.
テキストファイル	qr text file (2)	ファイルパス名
4D View	qr 4D View area (3)	N.A.
4D Chart	qr 4D Chart area (4)	N.A.
HTMLファイル	qr HTML file (5)	HTMLファイルパス名

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR Get document property

QR Get document property (area ; property) -> 戻り値

引数	型		説明
area	倍長整数	→	エリア参照
property	倍長整数	→	1 = 印刷ダイアログ, 2 = ドキュメント単位
戻り値	倍長整数	←	プロパティ値

説明

QR Get document property コマンドを使用し、印刷ダイアログの表示の有無、またはareaに表示されるドキュメントの単位を取得することができます。

propertyには、定数テーマ内にある次の定数を渡すことができます:

定数	型	値	コメント
qr printing dialog	倍長整数	1	プリントダイアログボックスの表示 • 値が1の場合、印刷の前に印刷ダイアログが表示されます。(デフォルト) • 値が0の場合、印刷の前に印刷ダイアログが表示されません。
qr unit	倍長整数	2	ドキュメントの単位 • 値が0の場合、ドキュメントの単位はポイントです。 • 値が1の場合、ドキュメントの単位はセンチです。 • 値が2の場合、ドキュメントの単位はインチです。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

無効なproperty引数を渡した場合、エラー番号-9852が生成されます。

QR Get drop column

QR Get drop column (area) -> 戻り値

引数	型	説明
area	倍長整数	→ エリア参照
戻り値	倍長整数	↩ ドロップされた値

説明

QR Get drop column コマンドは、ドロップ動作が行われた場所の値を返します。

- 戻り値が負の場合、カラム番号を示します (例えば、カラム番号3にドロップされた場合には-3)。
- 戻り値が正の場合、そのカラムの前にあるセパレータ上でドロップ動作が実行されたことを示します (例えば、カラム2の後ろにドロップされた場合には3)。ドロップ動作は、必ずしも既存のカラムの前で実行する必要はないことに留意してください。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR GET HEADER AND FOOTER

QR GET HEADER AND FOOTER (area ; selector ; leftTitle ; centerTitle ; rightTitle ; height {; picture {; pictAlignment}})

引数	型		説明
area	倍長整数	→	エリア参照
selector	倍長整数	→	1 = ヘッダ, 2 = フッタ
leftTitle	文字	←	左側に表示されるテキスト
centerTitle	文字	←	中央に表示されるテキスト
rightTitle	文字	←	右側に表示されるテキスト
height	倍長整数	←	ヘッダまたはフッタの高さ
picture	ピクチャー	←	表示するピクチャ
pictAlignment	倍長整数	←	ピクチャの整列属性

説明

QR GET HEADER AND FOOTER コマンドを使用し、ヘッダまたはフッタの内容とサイズを取得できます。

selector を使用して、ヘッダまたはフッタを選択します:

- selectorに1を指定すると、ヘッダ情報を取得できます。
- selectorに2を指定すると、フッタ情報を取得できます。

leftTitle, centerTitle そして rightTitleにはそれぞれ左側、中央、右側にあるヘッダまたはフッタの値が返されます。

heightには、そのレポートに対して選択された単位で表わされたヘッダまたはフッタの高さが返されます。

pictureには、ヘッダまたはフッタに表示されるピクチャが返されます。

pictAlignmentには、ヘッダまたはフッタに表示されるピクチャの整列属性が返されます。

- pictAlignmentが0の場合、そのピクチャは左揃えです。
- pictAlignmentが1の場合、そのピクチャは中央揃えです。
- pictAlignmentが2の場合、そのピクチャは右揃えです。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なselector引数を渡した場合、エラー番号-9852が生成されます。

例題

次のコードは、ヘッダタイトルの値とヘッダサイズを取得し、それを警告として表示します:

```
QR GET HEADER AND FOOTER(MyArea;1;$LeftText;$CenterText;$RightText;$height)
Case of
   :($LeftText #"")
      ALERT("The left title is "+Char(34)+$LeftText+Char(34))
   :($CenterText #"")
      ALERT("The center title is "+Char(34)+$CenterText+Char(34))
   :($RightText #"")
      ALERT("The right title is "+Char(34)+$RightText+Char(34))
   Else
      ALERT("No header title in this report.")
End case
ALERT("The height of the header is "+String($height))
```

QR Get HTML template

QR Get HTML template (area) -> 戻り値

引数	型		説明
area	倍長整数	→	エリア参照
戻り値	テキスト	↩	テンプレートとして使用されるHTMLコード

説明

QR Get HTML template コマンドは、クイックレポートareaに現在使用されているHTMLテンプレートを返します。戻り値はテキスト値であり、HTMLテンプレートの全内容が納められます。

テンプレートが特に指定されていない場合、デフォルトのテンプレートが返されます。手動またはプログラムにより出力先をHTMLファイルに設定されていない場合、テンプレートが返されない点に注意してください。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR GET INFO COLUMN

QR GET INFO COLUMN (area ; colNum ; title ; object ; hide ; size ; repeatedValue ; displayFormat)

引数	型		説明
area	倍長整数	→	エリア参照
colNum	倍長整数	→	カラム番号
title	文字	←	カラムタイトル
object	フィールド, 変数	←	カラムに割り当てられたオブジェクト
hide	倍長整数	←	0 = 表示, 1 = 非表示
size	倍長整数	←	カラムサイズ
repeatedValue	倍長整数	←	0 = 繰り返ししない, 1 = 繰り返す
displayFormat	テキスト	←	データの表示フォーマット

説明

リストモード

QR GET INFO COLUMN コマンドを使用して、既存のカラムに関するパラメタを取得することができます。

areaには、クイックレポートエリアの参照を指定します。

colNumには、パラメタを取得するカラムの番号を指定します。

titleには、カラムのヘッダに表示されるタイトルが返されます。

object には、そのカラムの実際のオブジェクト (変数、フィールド名、またはフォーミュラ) が返されます。

hideにはカラムが表示されるかされないかが返されます。

- hideが1の場合、カラムは非表示です。
- hideが0の場合、カラムは表示されます。

sizeには、カラムサイズがピクセル単位で返されます。値が負数の場合、サイズは自動に設定されています。

repeatedValueには、同一値印刷の有無が返されます。これはフィールドまたは変数の値が連続するレコード間で同一値であった場合、同じ値を印刷するかしないかを表します。

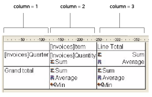
- repeatedValueが0の場合、同一値は印刷されません。
- repeatedValueが1の場合、同一値を印刷します。

formatには表示フォーマットが返されます。この表示フォーマットは、表示されるデータに対応した4Dフォーマットです。

クロステーブルモード

QR GET INFO COLUMN コマンドを使用して、同様のパラメータを取得できますが、取得しようとするパラメタにより、適用するエリアの参照が異なります。第一に、このコマンドをクロステーブルモードで使用した場合、引数title, hide, そして repeatedValueは意味を持ちません。取得したい値がカラムサイズ、データソース、表示フォーマットのいずれであるかによって、colNumに使用する値が変わります。

- カラムサイズ
  これは"視覚的"な属性であり、下図のようにカラムは左から右へと番号が付けられています:

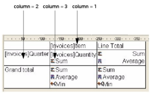

column = 1
column = 2
column = 3
50
100
150
200
250
300
350
[Invoices]Item
Line Total
[Invoices]Quarter
[Invoices]Quantity
Sum
Sum
Average
Grand total
Sum
Average
Min
Sum
Average
Min

次のコードは、クロステーブルレポートのすべてのカラムに対してサイズを自動に設定し、その他の要素は以前のまま変更しません:

```
For($i;1;3)
   QR GET INFO COLUMN(qr_area;$i;$title;$obj;$hide;$size;$rep;$format)
   QR SET INFO COLUMN(qr_area;$i;$title;$obj;$hide;0;$rep;$format)
End for
```

お分かりのように、カラムサイズだけを変更したいため、QR GET INFO COLUMNを使用してカラムのプロパティを取得し、それをQR GET INFO COLUMNに渡してカラムサイズ以外の項目は変更していません。

- データソース (オブジェクト) と表示フォーマット
  この場合、カラムの番号は次の図のように付けられます:


column = 2
column = 3
column = 1
50
100
150
200
250
300
350
[Invoices]Item
Line Total
[Invoices]Quarter
[Invoices]Quantity
Sum
Sum
Average
Grand total
Sum
Average
Min
Sum
Average
Min

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なcolNum引数を渡した場合、エラー番号-9852が生成されます。

QR Get info row

QR Get info row (area ; row) -> 戻り値

引数	型		説明
area	倍長整数	→	エリア参照
row	倍長整数	→	指定する行
戻り値	倍長整数	↩	0 = 表示, 1 = 非表示

説明

QR Get info row コマンドは、rowに渡した行に関する表示の有無を取得します。

rowには、情報を取得したい行を指定します:

- 正の整数である場合、表示属性を取得する小計 (ブレークレベル) を示します。
- テーマの定数を使用し、行アイテムを指定することができます

定数	型	値	コメント
qr detail	倍長整数	-2	レポートの詳細エリア
qr grand total	倍長整数	-3	総計エリア
qr title	倍長整数	-1	レポートタイトル

戻り値として行が表示されるか非表示であるかを示す値が返されます。戻り値が1の場合行は非表示に設定され、0の場合行は表示に設定されています。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なrow引数を渡した場合、エラー番号-9852が生成されます。

QR Get report kind

QR Get report kind (area) -> 戻り値

引数	型		説明
area	倍長整数	→	エリア参照
戻り値	倍長整数	↩	レポートタイプ

説明

QR Get report kind コマンドは、areaに渡したエリアのレポートタイプを取得します。

- 戻り値が1の場合、レポートはリストタイプです。
- 戻り値が2の場合、レポートはクロステーブルタイプです。

また、この関数の戻り値をテーマの定数と比較することもできます:

定数	型	値
qr cross report	倍長整数	2
qr list report	倍長整数	1

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR Get report table

QR Get report table (area) -> 戻り値

引数	型		説明
area	倍長整数	→	エリア参照
戻り値	倍長整数	←	テーブル番号

説明

QR Get report table コマンドは、areaに渡した参照のレポートエリア用のカレントテーブル番号を返します。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR GET SELECTION

QR GET SELECTION (area ; left ; top {; right {; bottom}})

引数	型		説明
area	倍長整数	→	エリア参照
left	倍長整数	←	左境界
top	倍長整数	←	上境界
right	倍長整数	←	右境界
bottom	倍長整数	←	下境界

説明

QR GET SELECTION コマンドは、選択されたセルの座標を返します。

left には、選択範囲の左側境界となるカラムの番号が返されます。left が0の場合、行全体が選択されています。

topには、選択範囲の上側境界となる行番号が返されます。topが0の場合、カラム全体が選択されています。

Note: leftとtopの両方が0の場合、エリア全体が反転表示（選択）されています。

rightには、選択範囲の右側境界となるカラムの番号が返されます。

bottomには、選択範囲の下側境界となる行番号が返されます。

Note: 何も選択されていない場合、left、top、right、bottomには-1が代入されます。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR GET SORTS

QR GET SORTS (area ; aColumns ; aOrders)

引数	型		説明
area	倍長整数	→	エリア参照
aColumns	実数配列	←	ソートされたカラム
aOrders	実数配列	←	ソート順

説明

QR GET SORTS コマンドは、次の2つの配列を作成します:

- aColumns
  この配列には、ソート順が設定されているすべてのカラムが格納されます。
- aOrders
  この配列の各要素には、対応するカラムの並び替え順が格納されます。
  - aOrders{$i}が1の場合、並び替え順は昇順です。
  - aOrders{$i}が-1の場合、並び替え順は降順です。

クロステーブルモード

クロステーブルモードの場合、結果の配列に格納される項目は2つ以下となります。これは、ソートが実行されるのが、カラム (1) と行 (2) だけであるためです (aColumnsの値)。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR Get text property

QR Get text property (area ; colNum ; rowNum ; property) -> 戻り値

引数	型		説明
area	倍長整数	→	エリア参照
colNum	倍長整数	→	カラム番号
rowNum	倍長整数	→	行番号
property	倍長整数	→	セルのプロパティ番号
戻り値	倍長整数	↩	選択したプロパティの値

説明

QR Get text property コマンドは、colNumとrowNumで指定されたセルのテキスト属性のプロパティ値を返します。

areaにはクイックレポートエリアの参照を渡します。

colNumにはセルのカラム番号を渡します。

rowNumにはセルの行の参照番号を渡します。

- 整数を渡して小計 (ブレーク) レベルを指定する
- テーマの定数を渡す

定数	型	値	コメント
qr detail	倍長整数	-2	レポートの詳細エリア
qr footer	倍長整数	-5	ページフッタ
qr grand total	倍長整数	-3	総計エリア
qr header	倍長整数	-4	ページヘッダ
qr title	倍長整数	-1	レポートタイトル

Note: rowNumに-4または-5を渡す場合、使用されなくてもcolNumを渡す必要があります。

Note: クロステーブルモードでは、行の値が常に正数であることを除き、原則は同じです。

property には、取得するテキスト属性の値または定数を渡します。テーマの定数を使用し、次の値を設定することができます:

定数	型	値	コメント
qr alternate background color	倍長整数	9	代替背景色
qr background color	倍長整数	8	背景色番号
qr bold	倍長整数	3	太字スタイル属性 (0 または 1)
qr font	倍長整数	1	Font numberが返すのと同様のフォント番号
qr font size	倍長整数	2	ポイント単位のフォントサイズ (9 ~ 255)
qr italic	倍長整数	4	イタリックスタイル属性 (0 または1)
qr justification	倍長整数	7	テキスト整列属性 (0 = デフォルト, 1 = 左, 2 = 中央, 3 = 右)
qr text color	倍長整数	6	フォントカラー属性 (カラー番号)
qr underline	倍長整数	5	下線スタイル属性 (0 または1)

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なcolNum引数を渡した場合、エラー番号-9852が生成されます。
無効なrowNum引数を渡した場合、エラー番号-9853が生成されます。
無効なproperty引数を渡した場合、エラー番号-9854が生成されます。

QR GET TOTALS DATA

QR GET TOTALS DATA (area ; colNum ; breakNum ; operator ; text)

引数	型		説明
area	倍長整数	⇨	エリア参照
colNum	倍長整数	⇨	カラム番号
breakNum	倍長整数	⇨	ブレーク番号
operator	倍長整数	⇦	セルの演算名
text	文字	⇦	セルの内容

説明

リストモード

QR GET TOTALS DATA コマンドを使用し、指定するブレークに関する詳細を取得できます。

areaにはクイックレポートエリアの参照を渡します。

colNumにはデータを取得するカラムの番号を渡します。

breakNumには、データを取得するブレーク番号 (小計または総計) を渡します。
- 小計：1から小計/ソート項目数までの数字
- 総計：-3または定数qr grand total (テーマ)

operatorには、セル内に存在するすべての演算子の和が返されます。テーマの定数を使用して値を処理できます:

定数	型	値
qr average	倍長整数	2
qr count	倍長整数	16
qr max	倍長整数	8
qr min	倍長整数	4
qr standard deviation	倍長整数	32
qr sum	倍長整数	1

返された値が0の場合、演算子は存在しません。

textにはセル内のテキストが返されます。

Note: operatorとtextは相互に排他的な引数です、したがってoperatorまたはtextのいずれか一方にしか結果が返されません。

クロステーブルモード

QR GET TOTALS DATA コマンドを使用し、指定したセルの詳細を取得できます。

areaにはクイックレポートエリアの参照を渡します。

colNumには、データを取得するセルのカラム番号を渡します。

breakNumには、データを取得するセルの行番号を渡します。

operatorには、セル内に存在するすべての演算子の和が返されます。テーマの定数を使用し、返された値を処理することができます (上記参照)。

textにはセル内のテキストが返されます。

次の図は、クロステーブルモードでのcolNumとbreakNumの組み合わせ方について示しています:

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なcolNum引数を渡した場合、エラー番号-9852が生成されます。
無効なbreakNum引数を渡した場合、エラー番号-9853が生成されます。

QR GET TOTALS SPACING

QR GET TOTALS SPACING (area ; subtotal ; value)

引数	型		説明
area	倍長整数	⇒	エリア参照
subtotal	倍長整数	⇒	小計番号
value	倍長整数	⇐	0=スペースなし, 32000=改ページ挿入, >0=ブレークレベルの上に追加するスペース, <0=比率指定

説明

QR GET TOTALS SPACING コマンドを使用し、小計行の上部の行間を取得することができます。このコマンドはリストモードにのみ適用されます。

areaにはクイックレポートエリアの参照を渡します。

subtotalには適用される小計レベル (またはブレークレベル) を渡します。subtotalは、1から小計/ソート数までの値となります。

valueは行間を示します:

- valueが0の場合、行間は追加されません。
- valueが32000の場合、改ページが挿入されます。
- valueが正の値の場合、間隔を表わす値をピクセル単位で示します。
- valueが負の値の場合、間隔を表わす値を小計行との比率で示します。例えば、-100であれば小計行の上に100%の間隔が設定されます。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なsubtotal引数を渡した場合、エラー番号-9852が生成されます。

QR INSERT COLUMN

QR INSERT COLUMN (area ; colNumber ; object)

引数	型		説明
area	倍長整数	⇒	エリア参照
colNumber	倍長整数	⇒	カラム番号
object	フィールド, 変数, ポインター	⇒	カラムに挿入するオブジェクト

説明

QR INSERT COLUMN コマンドは指定された位置にカラムの作成または挿入を行います。挿入された場所の右にあるカラムはすべて右側へ移動します。

colNumには、左から右へ順に付けられたカラム番号を渡します。

カラムのデフォルトのタイトルはobjectに渡された値です。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

例題

次のコードはクイックレポートエリアの一番目にカラムを挿入（または作成）し、カラムタイトルに"Field1"を設定し（デフォルト動作）、Field1の値をカラムの内容として代入します。

```
 QR INSERT COLUMN(MyArea;1;->[Table1]Field1)
```

QR New offscreen area

QR New offscreen area -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	作られたエリアの参照

説明

QR New offscreen area コマンドは、新しくクイックレポートのオフスクリーンエリアを作成し、その参照番号を返します。

QR ON COMMAND

QR ON COMMAND (area ; methodName)

引数	型		説明
area	倍長整数	→	エリア参照
methodName	文字	→	置き換えメソッド名

説明

QR ON COMMAND コマンドは、ユーザによるメニューコマンドの選択やボタンのクリックなどで、クイックレポートコマンドが起動されると、methodName に渡された4Dメソッドを実行します。areaが0の場合、データベースが閉じられるか、QR ON COMMAND (0;"")という構文で QR ON COMMAND コマンドが呼び出されるまで、methodNameが各クイックレポートエリアに適用されます。

methodNameは次の2つの引数を受け取ります:

- $1はエリアの参照(倍長整数)です。
- $2は選択されたコマンドのコマンド番号(倍長整数)です。この値は QR Commands テーマ内にある定数と比較可能です。

注: データベースをコンパイルする予定がある場合、$1と$2を使用しない場合でも、それぞれを倍長整数として定義する必要があります。

最初のコマンドを実行させたい場合には、呼び出されるメソッドに次の命令を記述する必要があります: QR EXECUTE COMMAND($1;$2)

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

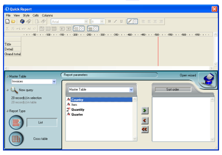
QR REPORT

QR REPORT ({aTable ;} document {; hierarchical {; wizard {; search {; methodName {; *}}}})

引数	型		説明
aTable	テーブル	⇨	レポートを作成するテーブル、省略時はデフォルトテーブル
document	文字	⇨	ロードするクイックレポートドキュメント
hierarchical	ブール	⇨	True = リレートするnテーブルを表示 Falseまたは省略 = 表示しない (デフォルト)
wizard	ブール	⇨	True = ウィザード・ボタンを表示 Falseまたは省略 = 表示しない (デフォルト)
search	ブール	⇨	True = 検索ツールとマスターテーブル選択を表示する Falseまたは省略 = 表示しない (デフォルト)
methodName	文字	⇨	Name of method to call
*	演算子	⇨	プリントダイアログボックスを表示しない

説明

QR REPORTは次に示すクイックレポートエディターを用いて作成された、aTableのレポートを印刷します。



クイックレポートエディターを使用すると、ユーザは独自のレポートを作成することができます。「クイックレポート」エディターを用いたレポートの作成に関する詳細は、4D *Design Reference* マニュアルを参照してください。

注:

- aTableが"非表示"に定義されている場合、エディターは表示されません。
- **QR REPORT**コマンドによって表示されたクイックレポートエディターには、リレーションの自動とマニュアルを切り替える**すべてのリレートを自動にする**チェックボックスがありません。デベロッパーはSET AUTOMATIC RELATIONSおよびSET FIELD RELATIONコマンドを使用して、リレートの状態を制御することができます。
- エディターは外部ウィンドウ内に呼び出され、このコンテキストでQR ON COMMANDコマンドを使用することはできません。しかしながら、インターフェースコマンドが有効化されている際には、methodName 引数を使用してカスタムコードを実行することができます。

document引数には、クイックレポートエディターを用いて作成され、ディスク上に保存されたレポートドキュメントを渡します。このドキュメントにはレポートに関する仕様が納められ、印刷されるレコードは含みません。
documentに空の文字列 ("") が指定されると、**QR REPORT**はファイルを開くダイアログボックスを表示し、ユーザは印刷するレポートを選択することができます。
document引数にドキュメントが指定され、そのドキュメントが存在しない場合（例えばdocumentに**Char(1)**を指定）、クイックレポートエディターが表示されます。

hierarchical引数には、フィールド選択リストにnテーブルを表示するかどうかを指定します。デフォルトでこの値はFalse (nテーブルを表示しない) に設定されています。

wizard引数には、クイックレポートエディター上にウィザードを開くボタンを表示するかどうかを指定し、結果としてウィザードへのアクセスを許可または禁止します。デフォルトでこの値はFalse (ウィザードへのアクセス不可) に設定されます。

search引数には、クイックレポートエディター上に新規クエリボタンとマス ターテーブルドロップダウンメニューを表示するかどうかを指定し、結果としてカレントテーブルとカレントマスターテーブルの変更を許可または禁止します。 デフォルトでこの値はFalse (検索ツールやマスターテーブルへアクセス不可) に設定されます。

methodName 引数は、メニューから選択、またはボタンをクリックすることによってQuick Reportエディターを呼び出すたびに実行される4D プロジェクトメソッドを指定します。この引数を使用することはQuick Reportエディターウィンドウのコンテキストにおいて QR ON COMMAND を使用することと同等の効果があります(QR ON COMMAND は範囲内のエリアのコンテキストにしか効果がありません)。具体的には、この新しい引数を使用してQuick Reportにて使用される文字コードを変更することが可能です。
methodName メソッドは二つの引数を受け取ることが可能です:

- *$1* にはエリア参照が格納されます(倍長整数)。
- *$2* には選択したコマンドの数が格納されます(倍長整数)。この値は QR Commands テーマ内の定数と比較可能です。

注: データベースをコンパイラーを使用してコンパイルしたい場合、使わない場合でも引数 $1 と $2 を明示的に倍長整数だと宣言する必要があります。

ユーザーによって選択された先頭のコマンドを実行したい場合、以下のコードを methodName メソッド内で使用して下さい:

```
QR EXECUTE COMMAND($1;$2).
```

methodName 引数に空の文字列("")が渡されるか省略された場合、メソッドは何も呼び出されず、**QR REPORT** の標準オペレーションが適用されます。

引数 * を指定しない場合、レポートの選択後に印刷ダイアログが表示されます。この引数を指定すると、これらのダイアログボックスは表示されず、レポートが印刷されます。

クイックレポートエディターが起動されない場合、レポートが印刷されるとシステム変数OKには1が、印刷されない場合 (つまり、ユーザーが印刷ダイアログで**キャンセル**をクリックした場合) には0が代入されます。

4D Server: このコマンドは、ストアドプロシージャーのコンテキストで、4D Server上で実行することができます。この場合次の制約があります:

- サーバーマシン上にはダイアログボックスを一切表示しないでください（特定の必要性がある場合を除く）。これを実現するには、コマンドを呼び出す際に、引数 * を指定する必要があります。
- 4D Serverでは、レポートエディターを表示させる構文は動作しません。これを行った場合、システム変数OKは0に設定されます。
- プリンター関連の問題が発生しても（用紙切れ、プリンター接続切断など）、エラーメッセージは生成されません。

例題 1

次の例では、ユーザが[People]テーブルを検索した後に、"Detailed Listing"というレポートが自動的に印刷されます:

```
QUERY([People])
If(OK=1)
   QR REPORT([People];"Detailed Listing";False;False;False;*)
End if
```

例題 2

次の例題では、ユーザは[People]テーブルを検索した後に、印刷するレポートを選択することができます:

```
QUERY([People])
If(OK=1)
   QR REPORT([People];"";False;False;False)
End if
```

例題 3

次の例題では、ユーザが[People]テーブルを検索した後にクイックレポートエディタが表示され、レポートの設計、保存、ロード、印刷を行うことができます:

```
QUERY([People])
If(OK=1)
   QR REPORT([People];Char(1);False;True)
End if
```

例題 4

SET FIELD RELATION コマンドの例題参照

例題 5

QR REPORT コマンドを使用して呼び出されたQuick Reportにて使用される文字コードを Mac Romanに変更したい場合:

```
QR REPORT([MyTable];Char(1);False;False;False;"myCallbackMeth")
```

レポートが生成される際、myCallbackMeth メソッドによってレポートを変換します:

```
C_LONGINT($1;$2)
If($2=qr cmd generate) //レポートが生成されたとき
   C_BLOB($myblob)
   C_TEXT($path;$text)
   QR EXECUTE COMMAND($1;$2) //コマンドの実行
   QR GET DESTINATION($1;$type;$path) //型の取得
   DOCUMENT TO BLOB($path;$myblob)
  //UTF-8を使用してテキストへと変更
   $text:=Convert to text($myblob;"UTF-8")
  //MacRoman 文字列を使用
   CONVERT FROM TEXT($text;"MacRoman";$myblob)
  //変換されたレポートを返す
   BLOB TO DOCUMENT($path;$myblob)
Else //そうでなければ、コマンドを実行
   QR EXECUTE COMMAND($1;$2)
End if
```

QR REPORT TO BLOB

QR REPORT TO BLOB (area ; BLOB)

引数	型		説明
area	倍長整数	→	エリア参照
BLOB	BLOB	←	クイックレポートを納めるBLOB

説明

QR REPORT TO BLOB コマンドは、areaに渡された参照番号のレポートをBLOB (変数またはフィールド) に格納します。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

例題

次のコードは、MyAreaに格納されているクイックレポートをBLOBフィールドに代入します。

```
QR REPORT TO BLOB(MyArea;[Table1]Field4)
```

QR RUN

QR RUN (area)

引数	型		説明
area	倍長整数	→	実行するエリアの参照

説明

QR RUNコマンドは、出力先を含め、クイックレポートの現在の設定を使用して、area引数に渡された参照番号のレポートエリアを実行します。QR SET DESTINATIONコマンドで出力タイプを変更できます。

属するエリアのテーブルでレポートが実行されます。areaがオフスクリーンエリアを指す場合、QR SET REPORT TABLEコマンドで使用するテーブルを指定する必要があります。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR SET AREA PROPERTY

QR SET AREA PROPERTY (area ; property ; value)

引数	型	説明
area	倍長整数	➡ エリア参照
property	倍長整数	➡ 指定するインタフェース要素
value	倍長整数	➡ 1 = 表示, 0 = 非表示

説明

QR SET AREA PROPERTYコマンドを使用し、propertyに渡したインタフェース要素 (ツールバーやメニューバー) を表示、または非表示に設定できます。

メニューバーおよびツールバーには1から6 (上から下) までの番号が振られ、値7はコンテキストメニュー専用です。
テーマの定数を使用して、インタフェース要素を指定することができます:

定数	型	値	コメント
qr view color toolbar	倍長整数	5	カラーツールバーのステータス取得 (表示=1, 非表示=0)
qr view column toolbar	倍長整数	6	カラムツールバーのステータス取得 (表示=1, 非表示=0)
qr view contextual menus	倍長整数	7	コンテキストメニューのステータス取得 (表示=1, 非表示=0)
qr view menubar	倍長整数	1	メニューバーのステータス取得 (表示=1, 非表示=0)
qr view operators toolbar	倍長整数	4	関数ツールバーのステータス取得 (表示=1, 非表示=0)
qr view standard toolbar	倍長整数	2	標準ツールバーのステータス取得 (表示=1, 非表示=0)
qr view style toolbar	倍長整数	3	スタイルツールバーのステータス取得 (表示=1, 非表示=0)

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なproperty引数を渡した場合、エラー番号-9852が生成されます。

QR SET BORDERS

QR SET BORDERS (area ; column ; row ; border ; row | level {; color})

引数	型		説明
area	倍長整数	⇨	エリア参照
column	倍長整数	⇨	カラム番号
row	倍長整数	⇨	行番号
border	倍長整数	⇨	罫線の合成値
row | level	倍長整数	⇨	線の太さ
color	倍長整数	⇨	罫線のカラー

説明

QR SET BORDERS コマンドを使用し、指定したセルの罫線スタイルを設定できます。

areaにはクイックレポートエリアの参照を渡します。

columnにはセルのカラム番号を渡します。

rowにはセルの行番号を渡します。以下を渡します:

- 正の整数の場合、その数値は小計 (ブレーク) レベルを示します。
- の定数

定数	型	値	コメント
qr detail	倍長整数	-2	レポートの詳細エリア
qr grand total	倍長整数	-3	総計エリア
qr title	倍長整数	-1	レポートタイトル

borderには、適用するセルの罫線を示す合計値を渡します。テーマの定数を使用できます:

定数	型	値	コメント
qr bottom border	倍長整数	8	下罫線
qr inside horizontal border	倍長整数	32	内側縦罫線
qr inside vertical border	倍長整数	16	内側横罫線
qr left border	倍長整数	1	左罫線
qr right border	倍長整数	4	右罫線
qr top border	倍長整数	2	上罫線

同時に複数の罫線を指定するために、borderに複数の値の合計値を渡すことができます。例えばborderに値5を渡すと、右側と左側の罫線に適用されます。

lineにはその線の太さを返します:

- 0 = 線なし
- 1 = 1/4ポイント
- 2 = 1/2ポイント
- 3 = 1ポイント
- 4 = 2ポイント

colorはその線の色です:

- colorが正の値である場合、指定の色を示します。
- colorが0の場合、黒色を示します。
- colorが-1の場合、色は変更されません。

Note: デフォルトの色は黒です。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なcolumn引数を渡した場合、エラー番号-9852が生成されます。
無効なrow引数を渡した場合、エラー番号-9853が生成されます。
無効なborder引数を渡した場合、エラー番号-9854が生成されます。
無効なline引数を渡した場合、エラー番号-9855が生成されます。

QR SET DESTINATION

QR SET DESTINATION (area ; type {; specifics})

引数	型		説明
area	倍長整数	⇒	エリア参照
type	倍長整数	⇒	レポートの出力先
specifics	文字, 変数	⇒	出力先の詳細

説明

QR SET DESTINATION コマンドは、areaに渡された参照番号のエリア用のレポート出力タイプを設定します。

type引数には、テーマの定数のいずれかを渡します。specifics引数の内容はtypeの値に基づきます。次の表はtypeおよびspecificsの両引数へ渡すことができる値を示しています:

定数	型	値	コメント
_O_qr 4D Chart area	倍長整数	4	*** 廃止予定の定数 ***
qr 4D View area	倍長整数	3	N.A.
qr HTML file	倍長整数	5	ファイルへのパス名
qr printer	倍長整数	1	N.A.
qr text file	倍長整数	2	ファイルへのパス名

qr text file (2): specifics引数に空の文字列を渡した場合、ファイル保存ダイアログが表示されます。それ以外の場合、ファイルはパス名で指定された場所に保存されます。
デフォルトのフィールド区切りはタブ記号 (文字コード9) です。また、デフォルトのレコード区切りはキャリッジリターン (文字コード13) です。区切り文字用の2つのシステム変数 (FldDelimitとRecDelimit) に値を指定して、これらのデフォルト値を変更できます。Windowsにおいて、FldDelimitが13である場合、このキャリッジリターンの後にラインフィード (文字コード10) が付加されます。これらの変数は、IMPORT TEXTなどの他のコマンドでも使用される点に注意してください。クイックレポートエディタのためにこれらの変数の値を変更すると、その変更はアプリケーションのあらゆる箇所に反映されます。

qr 4D View area (3): 4D Viewがユーザから使用可能である場合、4D View外部ウインドウが作成され、クイックレポートエリアの現在の設定を使用した結果が納められます。

qr HTML file (5): QR SET HTML TEMPLATEで設定されたテンプレートを使用して、HTMLファイルが作成されます。この変換の実行方法については、4D Design Referenceマニュアルを参照してください。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効な出力先typeを渡した場合、エラー番号-9852が生成されます。

例題

以下のコードは出力先をテキストファイルMydoc.txtに設定し、クイックレポートを実行します:

```
QR SET DESTINATION(MyArea;2;"MyDoc.txt")
QR RUN(MyArea)
```

QR SET DOCUMENT PROPERTY

QR SET DOCUMENT PROPERTY (area ; property ; value)

引数	型	説明
area	倍長整数	→ エリア参照
property	倍長整数	→ 1 = 印刷ダイアログ, 2 = ドキュメントの単位
value	倍長整数	→ プロパティ値

説明

QR SET DOCUMENT PROPERTYコマンドを使用し、印刷ダイアログの表示の有無、またはドキュメントで使用する単位の指定を行うことができます。

propertyには、定数テーマ内にある次の定数を渡すことができます:

定数	型	値	コメント
qr printing dialog	倍長整数	1	プリントダイアログボックスの表示 • 値が1の場合、印刷の前に印刷ダイアログが表示されます。(デフォルト) • 値が0の場合、印刷の前に印刷ダイアログが表示されません。
qr unit	倍長整数	2	ドキュメントの単位 • 値が0の場合、ドキュメントの単位はポイントです。 • 値が1の場合、ドキュメントの単位はセンチです。 • 値が2の場合、ドキュメントの単位はインチです。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なpropertyを渡した場合、エラー番号-9852が生成されます。

QR SET HEADER AND FOOTER

QR SET HEADER AND FOOTER (area ; selector ; leftTitle ; centerTitle ; rightTitle ; height {; picture {; pictAlignment}})

引数	型		説明
area	倍長整数	⇨	エリア参照
selector	倍長整数	⇨	1 = ヘッダ, 2 = フッタ
leftTitle	文字	⇨	左側に表示されるテキスト
centerTitle	文字	⇨	中央に表示されるテキスト
rightTitle	文字	⇨	右側に表示されるテキスト
height	倍長整数	⇨	ヘッダまたはフッタの高さ
picture	ピクチャー	⇨	表示するピクチャ
pictAlignment	倍長整数	⇨	ピクチャの整列属性

説明

QR SET HEADER AND FOOTER コマンドを使用し、ヘッダまたはフッタの内容とサイズを設定することができます。

selectorを使用して、ヘッダまたはフッタを選択します:

- selectorに1を指定すると、ヘッダ情報を設定できます。
- selectorに2を指定すると、フッタ情報を設定できます。

leftTitle, centerTitle および rightTitleにはそれぞれ、左側、中央、右側にあるヘッダまたはフッタの値を指定します。

heightには、そのレポートに対して選択した単位で表わされたヘッダまたはフッタの高さを指定します。

pictureには、ヘッダまたはフッタに表示されるピクチャを指定します。

pictAlignmentには、pictureに渡されたピクチャの整列属性を指定します。

- pictAlignmentが0の場合、そのピクチャは左揃えです。
- pictAlignmentが1の場合、そのピクチャは中央揃えです。
- pictAlignmentが2の場合、そのピクチャは右揃えです。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なselector引数を渡した場合、エラー番号-9852が生成されます。

例題

次のコードは、MyAreaのクイックレポートのヘッダタイトルとして"Center title"を設定し、ヘッダの高さを200ピクセルに設定します:

```
QR SET HEADER AND FOOTER(MyArea;1;"";"Center title";"";200)
```

QR SET HTML TEMPLATE

QR SET HTML TEMPLATE (area ; template)

引数	型		説明
area	倍長整数	⇒	エリア参照
template	テキスト	⇒	HTMLテンプレート

説明

QR SET HTML TEMPLATEコマンドは、クイックレポートエリアに使用されるHTMLテンプレートを設定します。テンプレートは、HTML形式でレポートを作成する際に使用されます。

テンプレートはデータを処理するのに、元のレポートに近いレイアウトを維持したり、独自のカスタムHTMLを適用するために、一連のタグを用います。

Note: 出力先をHTMLファイルに設定するため、まず初めにQR SET DESTINATIONコマンドを呼び出す必要がある点に注意してください。

HTMLタグ

<!--#4DQRheader--> ... <!--/#4DQRheader-->
これらのタグに挟まれたHTMLの内容を、カラムタイトルを元にして設定します。通常、これらのタグはレポートのタイトル行を定義するために使用します。

<!--#4DQRrow--> ... <!--/#4DQRrow-->
これらのタグに挟まれたHTMLの内容は、各データ行ごとに繰り返されます (詳細行と小計行を含む)。

<!--#4DQRcol--> ... <!--/#4DQRcol-->
これらのタグに挟まれたHTMLの内容は、1つの行の各データカラムごとに繰り返されます。カラムの順序は、レポート内の順序と同じです。<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->と一緒に使用した場合、<!--#4DQRcol--> ... <!--/#4DQRcol-->タグは、<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->を用いてその内容が挿入されたのではないカラムを対象とします。
例えば、5つのカラムから構成されるレポートの場合、<!--#4DQRcol;2--> ... <!--/#4DQRcol;2-->を使用して2番目のカラムからデータを挿入すると、<!--#4DQRcol--> ... <!--/#4DQRcol-->は、各行のカラム1、3、4、5を処理します。後者のタグは、<!--#4DQRcol;2--> ... <!--/#4DQRcol;2-->を用いてその内容が作成されたカラムを無視します。

<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->
これらのタグに挟まれたHTMLの内容は、番号が"n"であるレポートカラムから取り出されます。
例えば、3つのカラムから構成されるレポートにHTML出力として異なるカラム順を表示したい場合、次のように指定することができます: <!--#4DQRrow--> <!--#4DQRcol;3--> ... <!--/#4DQRcol;3--><!--#4DQRcol;2--> ... <!--/#4DQRcol;2--><!--#4DQRcol;1--> ... <!--/#4DQRcol;1--> <!--/#4DQRrow-->

この例では、カラムはレポートと逆の順序で挿入されます。

<!--#4DQRfont--> ... <!--/#4DQRfont-->
これらのタグに挟まれたHTMLの内容は、カレントカラムまたはセルのフォントおよびフォントサイズとして割り当てられます。
<!--#4DQRfont-->は、HTMLフォント定義に置き換えられ、<!--/#4DQRfont-->は対応する終了タグ () に置き換えられます。

<!--#4DQRface--> ... <!--/#4DQRface-->
これらのタグに挟まれたHTMLの内容は、カレントカラムまたはセルのフォントスタイルとして割り当てられます。
<!--#4DQRface-->は、HTMLフェース定義に置き換えられ、<!--#4DQRface-->は対応する終了タグ (</face>) に置き換えられます。

<!--#4DQRbgcolor-->
このカラータグは、カレントセルのカレントカラーで置き換えられます。

<!--#4DQRdata-->
このタグは、カレントセルのカレントデータで置き換えられます。

<!--#4DQRlHeader--><!--#4DQRdata--><!--/#4DQRlHeader-->
<!--#4DQRcHeader--><!--#4DQRdata--><!--/#4DQRcHeader-->
<!--#4DQRrHeader--><!--#4DQRdata--><!--/#4DQRrHeader-->
これらのタグはそれぞれ、左ヘッダ、中央ヘッダ、右ヘッダのデータで置き換えられます。

<!--#4DQRlFooter--><!--#4DQRdata--><!--/#4DQRlFooter-->
<!--#4DQRcFooter--><!--#4DQRdata--><!--/#4DQRcFooter-->
<!--#4DQRrFooter--><!--#4DQRdata--><!--/#4DQRrFooter-->
これらのタグはそれぞれ、左フッタ、中央フッタ、右フッタのデータで置き換えられます。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR SET INFO COLUMN

QR SET INFO COLUMN (area ; colNum ; title ; object ; hide ; size ; repeatedValue ; displayFormat)

引数	型	説明
area	倍長整数	⇨ エリア参照
colNum	倍長整数	⇨ カラム番号
title	文字	⇨ カラムタイトル
object	フィールド, 変数	⇨ カラムに割り当てられたオブジェクト
hide	倍長整数	⇨ 0 = 表示, 1 = 非表示
size	倍長整数	⇨ カラムサイズ
repeatedValue	倍長整数	⇨ 0 = 繰り返ししない, 1 = 繰り返す
displayFormat	文字	⇨ データの表示フォーマット

説明

リストモード

QR SET INFO COLUMN コマンドを使用して、既存のカラムに関するパラメタを設定できます。

areaには、クイックレポートエリアの参照を指定します。

colNumには、修正するカラムの番号を指定します。

titleには、カラムのヘッダに表示されるタイトルを指定します。

objectには、そのカラムの実際のオブジェクト (変数、フィールド、またはフォーミュラ) を指定します。

hideには、カラムを表示するか、あるいは非表示にするかを指定します。

- 1を指定すると、カラムは非表示に設定されます。
- 0を指定すると、カラムは表示に設定されます。

sizeには、カラムに割り当てるサイズをピクセル単位で指定します。-1はカラムのサイズを自動設定にします。

repeatedValueには、同一値印刷の有無を指定します。これは、フィールドまたは変数の値が連続するレコード間で同一値であった場合、同じ値を印刷するか、しないかを表します。

- repeatedValueが0の場合、同一値は印刷されません。
- repeatedValueが1の場合、常に印刷します。

displayFormatには、表示フォーマットを指定します。この表示フォーマットは、表示されるデータに対応した4Dフォーマットです。

次のコードは、カラム番号1のタイトルとしてTitleをセットし、カラムの内容としてField2をセットし、幅150ピクセルでカラムを表示してフォーマットを###.##に設定します。

```
QR SET INFO COLUMN(area;1;"Title";"[Table1]Field2";0;150;0;"###,##")
```

クロステーブルモード

QR SET INFO COLUMN コマンドを使用して同様のパラメタを設定できますが、設定しようとするパラメタにより、適用するエリアの参照が異なります。
第一に、このコマンドをクロステーブルモードで用いた場合、引数title、hide、repeatedValueは使用されません。設定したい値がカラムサイズ、データソース、表示フォーマットのいずれであるかによって、colNumに使用する値が変わります。

- カラムサイズ
  これは"視覚的"な属性であり、下図のようにカラムは左から右へと番号が付けられています。

次のコードは、クロステーブルレポートのすべてのカラムに対してサイズを自動に設定し、その他の要素は以前のまま変更しません:

```
For($i;1;3)
   QR GET INFO COLUMN(qr_area;$i;$title;$obj;$hide;$size;$rep;$format)
   QR SET INFO COLUMN(qr_area;$i;$title;$obj;$hide;0;$rep;$format)
End for
```

カラムサイズだけを変更したいため、QR GET INFO COLUMNを使用してカラムのプロパティを取得し、それをQR SET INFO COLUMNに渡してカラムサイズ以外の項目は変更していません。

- データソース (オブジェクト) と表示フォーマット
  この場合、カラム番号は次の図のように作用します:

QR SET INFO COLUMNコマンドを使用しても、すべてのセルに対応できないことにお気付きでしょう。上図の中で番号が付けられていないセルに関しては、QR SET TOTALS DATAコマンドを用いて対処します。

次のコードは、基本的なクロステーブルレポートの作成に必要となる3つのセルに対し、データソースを割り当てます:

```
QR SET REPORT TABLE(qr_area;Table(->[Invoices]))
ALL RECORDS([Invoices])
QR SET REPORT KIND(qr_area;2)
QR SET INFO COLUMN(qr_area;1;"";->[Invoices]Item;1;-1;1;"")
QR SET INFO COLUMN(qr_area;2;"";->[Invoices]Quarter;1;-1;1;"")
QR SET INFO COLUMN(qr_area;3;"";->[Invoices]Quantity;1;-1;1;"")
```

この結果、レポートエリアは次のようになります:

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なcolNum引数を渡した場合、エラー番号-9852が生成されます。

QR SET INFO ROW

QR SET INFO ROW (area ; row ; hide)

引数	型		説明
area	倍長整数	⇒	エリアの参照
row	倍長整数	⇒	行指定
hide	倍長整数	⇒	0 = 表示, 1 = 非表示

説明

QR SET INFO ROW コマンドは、rowに渡した行を表示/非表示に設定します。

rowには、設定したい行を指定します:

- rowが正の整数である場合、表示属性を設定する小計 (ブレークレベル) を示します。
- テーマの定数を使用し、行アイテムを指定することができます

定数	型	値	コメント
qr detail	倍長整数	-2	レポートの詳細エリア
qr grand total	倍長整数	-3	総計エリア
qr title	倍長整数	-1	レポートタイトル

hideには、行を表示するか、または非表示にするかを示す値を指定します:

- hideが1の場合、行は非表示に設定されます。
- hideが0の場合、行は表示に設定されます。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なrow引数を渡した場合、エラー番号-9852が生成されます。

例題

次のコードは、詳細行を隠します:

```
QR SET INFO ROW(area;qr detail;1)
```

QR SET REPORT KIND

QR SET REPORT KIND (area ; type)

引数	型		説明
area	倍長整数	⇒	エリア参照
type	倍長整数	⇒	レポートタイプ

説明

QR SET REPORT KIND コマンドは、areaに渡した参照番号のエリアのレポートタイプを設定します。

- typeが1の場合、レポートはリストタイプです。
- typeが2の場合、レポートはクロステーブルタイプです。

またテーマの定数を使用することもできます:

定数	型	値
qr cross report	倍長整数	2
qr list report	倍長整数	1

既存のカレントレポートに対して新たにタイプを設定した場合、前回の設定は破棄され、新しい空のレポートが用意されます。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なtype引数を渡した場合、エラー番号-9852が生成されます。

QR SET REPORT TABLE

QR SET REPORT TABLE (area ; aTable)

引数	型		説明
area	倍長整数	⇨	エリア参照
aTable	倍長整数	⇨	テーブル番号

説明

QR SET REPORT TABLE コマンドは、areaに渡した参照のレポートエリアに、tableに渡した番号のテーブルをカレントテーブルとして設定します。

レポートにテーブルを割り当てることが大切な理由は、レポートエディタではそのテーブルのカレントセレクションを使用し、必要に応じてデータの表示や計算の実行、リレートの設定を行うためです。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なtable引数を渡した場合、エラー番号-9852が生成されます。

QR SET SELECTION

QR SET SELECTION (area ; left ; top ; right ; bottom)

引数	型		説明
area	倍長整数	⇒	エリア参照
left	倍長整数	⇒	左境界
top	倍長整数	⇒	上境界
right	倍長整数	⇒	右境界
bottom	倍長整数	⇒	下境界

説明

QR SET SELECTION コマンドを使用するとマウスクリックをした場合と同様に、セルや行、カラム、またはエリア全体を反転表示 (選択) することができます。また、現在選択されている範囲を解除することもできます。

leftには左側境界となるカラムの番号を指定します。leftが0の場合行全体が選択されます。
topには上側境界となる行番号を指定します。topが0の場合カラム全体が選択されます。
rightには右側境界となるカラムの番号を指定します。
bottomには下側境界となる行番号を指定します。

Notes:

- leftとtopの両方が0の場合、エリア全体が反転表示 (選択) されます。
- 選択範囲を設定したくない場合、left、top、right、bottomに-1を渡します。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR SET SORTS

QR SET SORTS (area ; aColumns {; aOrders})

引数	型		説明
area	倍長整数	⇨	エリア参照
aColumns	実数配列	⇨	カラム
aOrders	実数配列	⇨	ソート方向

説明

QR SET SORTS コマンド使用し、areaに渡した参照のレポート内にあるカラムのソート順を設定できます。

aColumnsには、ソート順を設定しようとする各カラムのカラム番号を格納します。

aOrdersには、aColumns配列内の対応するカラムのソート順を格納しなくてはなりません。

- aOrders{$i}が1の場合、ソート順は昇順です。
- aOrders{$i}が-1の場合、ソート順は降順です。

クロステーブルモード

クロステーブルモードの場合、2つより多くの項目を配列に納めることはできません。ソートされるのは、カラム (1) と行 (2) だけです。データ (カラムと行の交差する場所に位置する) をソートすることはできません。

次に示すコードは、クロステーブルレポートにおいて行だけを並び替えます:

```
ARRAY REAL($aColumns;1)
$aColumns{1}:=2
ARRAY REAL($aOrders;1)
$aOrders{1}:=-1 `Alphabetic sort for rows
QR SET SORTS(qr_area;$aColumns;$aOrders)
```

無効なarea番号を渡した場合、エラー番号-9850が生成されます。

QR SET TEXT PROPERTY

QR SET TEXT PROPERTY (area ; colNum ; rowNum ; property ; value)

引数	型		説明
area	倍長整数	⇨	エリア参照
colNum	倍長整数	⇨	カラム番号
rowNum	倍長整数	⇨	行番号
property	倍長整数	⇨	セルのプロパティ番号
value	倍長整数	⇨	選択したプロパティの値

説明

QR SET TEXT PROPERTYコマンドを使用し、colNumとrowNumで指定されたセルのテキスト属性を設定できます。

areaにはクイックレポートエリアの参照を渡します。

colNumにはセルのカラム番号を渡します。

rowNumにはセルの行の参照番号を渡します:

- 正の値である場合、対応する小計 (ブレークレベル) を示します。
- テーマの定数を使用し、行アイテムを指定することができます。

定数	型	値	コメント
qr detail	倍長整数	-2	レポートの詳細エリア
qr footer	倍長整数	-5	ページフッタ
qr grand total	倍長整数	-3	総計エリア
qr header	倍長整数	-4	ページヘッダ
qr title	倍長整数	-1	レポートタイトル

Note: rowNumに-4または-5を渡す場合、使用されなくてもcolNumを渡す必要があります。

Note: クロステーブルモードでは、行の値が常に正数であることを除き、原則は同じです。

propertyには、割り当てるテキスト属性の値または定数を渡します。テーマの定数を使用し、次の値を指定することができます:

定数	型	値	コメント
qr alternate background color	倍長整数	9	代替背景色
qr background color	倍長整数	8	背景色番号
qr bold	倍長整数	3	太字スタイル属性 (0 または 1)
qr font	倍長整数	1	Font numberが返すのと同様のフォント番号
qr font size	倍長整数	2	ポイント単位のフォントサイズ (9 ~ 255)
qr italic	倍長整数	4	イタリックスタイル属性 (0 または1)
qr justification	倍長整数	7	テキスト整列属性 (0 = デフォルト, 1 = 左, 2 = 中央, 3 = 右)
qr text color	倍長整数	6	フォントカラー属性 (カラー番号)
qr underline	倍長整数	5	下線スタイル属性 (0 または1)

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なcolNum引数を渡した場合、エラー番号-9852が生成されます。
無効なrowNum引数を渡した場合、エラー番号-9853が生成されます。
無効なproperty引数を渡した場合、エラー番号-9854が生成されます。

例題

このメソッドは、最初のカラムのタイトルに対して複数の属性を定義します:

```
  `The following call assigns the font Times:
 QR SET TEXT PROPERTY(qr_area;1;-1;qr font;Font number("Times"))
  `assigning the font size 10 points:
 QR SET TEXT PROPERTY(qr_area;1;-1;qr font size;10)
  `assigning the font attribute Bold:
 QR SET TEXT PROPERTY(qr_area;1;-1;qr bold;1)
  `assigning the font attribute Italic:
 QR SET TEXT PROPERTY(qr_area;1;-1;qr italic;1)
  `assigning the font attribute Underline:
 QR SET TEXT PROPERTY(qr_area;1;-1;qr underline;1)
  `assigning the color bright green:
 QR SET TEXT PROPERTY(qr_area;1;-1;qr text color;0x0000FF00)
```

QR SET TOTALS DATA

QR SET TOTALS DATA (area ; colNum ; breakNum ; operator | value)

引数	型		説明
area	倍長整数	→	エリア参照
colNum	倍長整数	→	カラム番号
breakNum	倍長整数	→	ブレーク番号
operator | value	倍長整数, 文字	→	セルの演算子の値またはセルの内容

説明

Note: このコマンドで小計を作成することはできません。

リストモード

QR SET TOTALS DATA コマンドを使用し、ブレーク (総計または小計) に関する詳細を設定できます。

areaにはクイックレポートエリアの参照を渡します。

colNumにはデータをセットすセルのカラム番号を渡します。

breakNumには、データをセットするブレークの番号 (小計または総計) を渡します。小計の場合、breaknumにはソート番号を指定します。総計の場合、breaknum には-3または定数qr grand totalを指定します。

operatorには、セル内に存在するすべての演算子の和を指定します。テーマの定数を使用して、<operator>を指定することができます:

定数	型	値
qr average	倍長整数	2
qr count	倍長整数	16
qr max	倍長整数	8
qr min	倍長整数	4
qr standard deviation	倍長整数	32
qr sum	倍長整数	1

operatorが0の場合、演算子は存在しません。

valueにはセルに格納するテキストを指定します。

Note: operatorとvalueは相互に排他的な引数です、したがってoperatorまたはvalueのいずれか一方だけを設定してください。

次の値を渡すことができます:
- # …ブレークまたは小計をトリガする値。
- ##S …合計に置き換えられます。
- ##A …平均に置き換えられます。
- ##C …回数に置き換えられます。
- ##X …最大値に置き換えられます。
- ##N …最小値に置き換えられます。
- ##D …標準偏差に置き換えられます。
- ##xx …xxにはカラム番号を指定し、その番号のカラムのフォーマットを用いて、カラムの値で置き換えられます。このカラムが存在しない場合、置き換えは実行されません。

クロステーブルモード

QR SET TOTALS DATA コマンドを使用し、特定のセルの詳細を設定することができます。

areaにはクイックレポートエリアの参照を渡します。

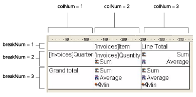
colNumには、データを設定するセルのカラム番号を渡します。

breakNumには、データを設定するセルの行番号を渡します。

operatorには、セル内に存在するすべての演算子の和を指定します。テーマの定数を使用できます (上記参照)。

valueにはセルに格納するテキストを指定します。

次の図は、クロステーブルモードでカラムとブレーク引数の組み合わせ方について示しています:


colNum = 1
colNum = 2
colNum = 3
breakNum = 1
breakNum = 2
breakNum = 3
50
100
150
200
250
300
350
[Invoices]Item
Line Total
[Invoices]Quarter
[Invoices]Quantity
Sum
Sum
Average
Grand total
Sum
Average
Min
Sum
Average
Min

Supported Types of Data

サポートされるデータのタイプ:

- タイトル
  タイトルは、引数valueを用いて渡します。実際のところvalueには文字列を指定し、次に示すセルに対してのみ渡すことができます。
  colNum=3 breakNum=1およびcolNum=1 breakNum=3
- 演算子
  単独の演算子、または演算子の組み合わせ (上図を参照) は、次のセルに対して渡すことができます:
  colNum=2, breakNum=2
  colNum=3, breakNum=2
  colNum=2, breakNum=3
  これら最後の2つの値は、セル (Column 3; Row 3) に対しても影響を与える点に注意してください。セル (Column 2; Row 3) に集計計算を定義する場合、セル (Column 2; Row 3) の内容は常にセル(Column 3; Row 3) をも定義します。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なcolNum引数を渡した場合、エラー番号-9852が生成されます。
無効なbreakNum引数を渡した場合、エラー番号-9853が生成されます。

QR SET TOTALS SPACING

QR SET TOTALS SPACING (area ; subtotal ; value)

引数	型	説明
area	倍長整数	⇒ エリア参照
subtotal	倍長整数	⇒ 小計番号
value	倍長整数	⇒ 0=スペースなし, 32000=改ページ挿入, >0=ブレークレベルの上に追加するスペース, <0=比率指定

説明

QR SET TOTALS SPACING コマンドを使用し、小計行の上部の行間を設定できます。このコマンドはリストモードにのみ適用されます。

area にはクイックレポートエリアの参照を渡します。
subtotal には適用される小計レベル (ブレークレベル) を渡します。

valueは行間を示します:

- valueが0の場合、行間は追加されません。
- valueが32000の場合、改ページが挿入されます。
- valueが正の値の場合、間隔を表わす値をピクセル単位で示します。
- valueが負の値の場合、間隔を表わす値を小計行との比率で示します。例えば、-100であれば小計行の上に100%の間隔が設定されます。

Note: 小計行の上部の行間により小計行が次のページへ押し出される場合、その小計行の上部には行間は追加されません。

無効なarea番号を渡した場合、エラー番号-9850が生成されます。
無効なsubtotal引数を渡した場合、エラー番号-9852が生成されます。

クエリ

- DESCRIBE QUERY EXECUTION
- Find in field
- Get last query path
- Get last query plan
- GET QUERY DESTINATION
- Get query limit
- ORDER BY
- ORDER BY FORMULA
- QUERY
- QUERY BY EXAMPLE
- QUERY BY FORMULA
- QUERY SELECTION
- QUERY SELECTION BY FORMULA
- QUERY SELECTION WITH ARRAY
- QUERY WITH ARRAY
- SET QUERY AND LOCK
- SET QUERY DESTINATION
- SET QUERY LIMIT

DESCRIBE QUERY EXECUTION

DESCRIBE QUERY EXECUTION (status)

引数	型		説明
status	ブール	⇒	True=内部クエリ分析を有効にする, False=内部クエリ分析を無効にする

説明

DESCRIBE QUERY EXECUTION コマンドはカレントプロセスにおいて、クエリの分析を有効にしたり無効にしたりします。このコマンドは、4DランゲージのQUERYコマンドのようなクエリの文脈でのみ機能します。

status 引数にTrueを設定してこのコマンドを呼び出すと、クエリ分析モードが有効になります。このモードでは、4Dエンジンはデータに対して行われるクエリごとに2つの情報を内部的に記録します:

- クエリが実行される直前の、クエリに関する詳細な説明。言い換えれば、実行しようとする計画 (クエリプラン)。
- 実際に実行されたクエリの詳細な説明 (クエリパス)。

記録される情報はクエリのタイプ (インデックス付き, シーケンシャル)、見つけたレコード数、実行するクエリ条件ごとに必要な時間を含みます。これらの情報はGet Last Query PlanとGet Last Query Pathコマンドで読みだすことができます。

通常、クエリプランの説明とクエリパスは同じです。しかしパフォーマンスを向上させるため、4Dはクエリ実行時に動的な最適化を行うことがあるため、これら2つが異なることもあります。例えば4Dエンジンがそのほうが早いと判断すれば、インデックス付きフィールドに対するクエリをシーケンシャルクエリに変更することがあります - これは特に検索対象のセレクションが少ないときに発生します。

status 引数にFalseを渡すとクエリの分析を停止します。クエリ分析モードはアプリケーションを遅くします。

例題

以下の例題は、コマンドを使用して取得できる情報のタイプを示します:

```
C_TEXT($vResultPlan;$vResultPath)
DESCRIBE QUERY EXECUTION(True) //analysis mode
QUERY([Employees];[Employees]LastName="T@";*) // Search for employees whose last name starts with T...
QUERY([Employees]; & ;[Companies]Name="H@";*) // that work for a company whose name starts with H
QUERY([Employees]; & ;[Employees]Salary>2500;*) // whose salary is > 2500
QUERY([Employees]; & ;[Cities]Pop<50000) // that live in a city with less than 50,000 inhabitants
$vResultPlan:=Get last query plan(Description in Text Format)
$vResultPath:=Get last query path(Description in Text Format)
DESCRIBE QUERY EXECUTION(False) //End of analysis mode
```

このコードを実行後、$vResultPlan と $vResultPath には実行されたクエリの定義が含まれます。例:

```
$vResultPlan :
   Employees.LastName == T@ And Employees.Salary > 2500 And Join on Table : Companies  :  Employees.Company =
Companies.Name [index : Companies.Name ] LIKE H@ And Join on Table : Cities  :  Employees.City = Cities.Name
[index : Cities.Pop ] < 50000
$vResultPath :
(Employees.LastName == T@ And Employees.Salary  > 2500) And (Join on Table : Companies  :  Employees.Company  =
Companies.Name with filter {[index : Companies.Name ] LIKE H@}) And (Join on Table : Cities  :  Employees.City =
Cities.Name with filter {[index : Cities.Pop ] < 50000})   (3 records found in 1 ms)
```

Description in XML Format 定数が Get last query path コマンドに渡されると、$vResultPath にはXMLで表現された定義が返されます:

```
$vResultPath :
   <QueryExecution>
      <steps description="And" time="0" recordsfounds="1227">
         <steps description="[Merge] : ACTORS with CITIES" time="13" recordsfounds="1227">
            <steps  description="[Join] : ACTORS.Birth_City_ID =CITIES.City_ID" time="13"  recordsfounds="1227"/>
         </steps>
      </steps>
   </QueryExecution>
```

Find in field

Find in field (targetField ; value) -> 戻り値

引数	型		説明
targetField	フィールド	→	検索を実行するフィールド
value	フィールド, 変数	→	検索する値
		←	検索された値
戻り値	倍長整数	↩	検索されたレコード番号 または レコードが検索されなかった場合、-1

説明

Find in field コマンドは、targetFieldフィールドの値がvalueに等しい最初のレコードのレコード番号を返します。何もレコードが見つからなければ、は-1を返します。

このコマンドを呼び出した後、valueには見つかった値が返されます。これにより、文字列フィールド上でワイルドカード ("@") を使って検索し、見つかった値を知る事ができます。

このコマンドは、カレントセレクションまたはカレントレコードを変更しません。

このコマンドは速く、特にレコード入力中に重複データの入力を防ぐのに役立ちます。

以前のバージョンについて: このコマンドは、初期のバージョンでの名称はFind index keyコマンドで、インデックスが付けられたフィールドのみを対象としていました。4D v11 SQLからこの制限はなくなり、そのためにコマンド名が変更されました。

例題 1

オーディオCD用のデータベースで、レコード入力中に既に登録されている歌手かどうかを確認したいとします。同姓同名も存在するため[Singer]Nameフィールドを重複不可にせず、入力フォームで[Singer]Nameフィールドのオブジェクトメソッドに下記のコードを書くことにします:

```
If(Form event=On Data Change)
   $RecNum:=Find in field([Singer]Name;[Singer]Name)
   If($RecNum #-1) ` この名前が既に入力済みなら
      CONFIRM("この名前の歌手が既に存在します。レコードを表示しますか?";"Yes";"No")
      If(OK=1)
         GOTO RECORD([Singer];$RecNum)
      End if
   End if
End if
```

例題 2

この例題は有効な値を持つレコードがあるかを調べます。

```
C_LONGINT($id;$1)
$id:=$1
If(Find in field([MyTable]MyID;$id)>=0)
   $0:=True
Else
   $0:=False
End if
```

備考: 演算子 >= が示しているのは、全てのレコード番号を対象としていることです。本コマンドはレコード番号を返しますが、レコード番号の最小値は0なので、全てのレコード番号が対象となります。

Get last query path

Get last query path (descFormat) -> 戻り値

引数	型		説明
descFormat	倍長整数	→	説明フォーマット (テキストまたはXML)
戻り値	文字	↩	最後に実行されたクエリパスの説明

説明

Get Last Query Path コマンドは、データに対して行われた最後のクエリの実際のパスの説明を返します。クエリの説明に関する詳細は、DESCRIBE QUERY EXECUTION コマンドの説明を参照してください。

descFormat 引数に渡した値に基づき、説明はテキストまたはXMLフォーマットで返されます。テーマの定数を使用できます:

定数	型	値
Description in Text Format	倍長整数	0
Description in XML Format	倍長整数	1

このコマンドを実行する前にDESCRIBE QUERY EXECUTION コマンドをセッション中で使用して、クエリ分析モードを有効にしなければなりません。

クエリパスの説明を (Get Last Query Plan コマンドで取得できる) クエリプランの説明と比較し、最適化を行うことができます。

Get last query plan

Get last query plan (descFormat) -> 戻り値

引数	型		説明
descFormat	倍長整数	→	説明フォーマット (テキストまたはXML)
戻り値	文字	↩	最後に実行されたクエリプランの説明

説明

Get Last Query Plan コマンドは、データに対して行われた最後のクエリのクエリプランの説明を返します。クエリの説明に関する詳細は、DESCRIBE QUERY EXECUTION コマンドの説明を参照してください。

descFormat 引数に渡した値に基づき、説明はテキストまたはXMLフォーマットで返されます。テーマの定数を使用できます:

定数	型	値
Description in Text Format	倍長整数	0
Description in XML Format	倍長整数	1

このコマンドを実行する前にDESCRIBE QUERY EXECUTION コマンドをセッション中で使用して、クエリ分析モードを有効にしなければなりません。

クエリプランの説明を (Get Last Query Path コマンドで取得できる) クエリパスの説明と比較し、最適化を行うことができます。

GET QUERY DESTINATION

GET QUERY DESTINATION (destinationType ; destinationObject ; destinationPtr)

引数	型		説明
destinationType	倍長整数	←	0 = カレントセレクション、1 = セット、2 = 命名セレクション、3 = 変数
destinationObject	文字	←	セット名、命名セレクション名、または空の文字列
destinationPtr	ポインター	←	destinationType=3のとき、ローカル変数へのポインター

説明

GET QUERY DESTINATIONコマンドは実行中のプロセスのクエリ結果の格納先を返します。デフォルトでクエリの結果はカレントセレクションとして反映されます。しかしこの動作はSET QUERY DESTINATIONコマンドを使用して変更できます。

destinationType 引数にはクエリの格納先を示す値が、そして destinationObject 引数には格納先の名前が返されます (指定されている場合)。destinationType に返される値はQueriesテーマの定数と比較できます:

定数	型	値
Into current selection	倍長整数	0
Into named selection	倍長整数	2
Into set	倍長整数	1
Into variable	倍長整数	3

destinationObject 引数に返される値は destinationType により異なります:

destinationType 引数	destinationObject 引数
0 (カレントセレクション)	destinationObject は空の文字列
1 (セット)	destinationObject にはセット名が返される
2 (命名セレクション)	destinationObject には命名セレクション名が返される
3 (変数)	destinationObject は空の文字列 (destinationPtr 引数を使用します)

クエリの格納先がローカル変数の場合 (destinationType = 3)、コマンドはこの変数へのポインターを destinationPtr 引数に返します。

例題

一時的にクエリの格納先を変更し、後で元に戻す:

```
GET QUERY DESTINATION($vType;$vName;$ptr)
 // 現在の設定値を取得
SET QUERY DESTINATION(Into set;"$temp")
 // 一時的に格納先を変更
QUERY(...) // 検索実行
SET QUERY DESTINATION($vType;$vName;$ptr)
 // 元の設定に戻す
```

Get query limit

Get query limit -> 戻り値

引数	型		説明
戻り値	倍長整数	←	クエリ結果の制限数値 0 = 制限なし

説明

Get query limitコマンドはカレントプロセスでクエリ結果として返されるレコードの上限値を返します。

この上限値はSET QUERY LIMITコマンドを使用して設定できます。

デフォルトでは制限がなく、この場合コマンドは0を返します。

ORDER BY

ORDER BY ({aTable ;}{ aField }{; > または < }{; aField2 ; > または <2 ; ... ; aFieldN ; > または <N}{; *})

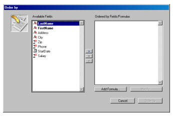
引数	型		説明
aTable	テーブル	⇒	セレクションをソートするテーブル, または 省略した場合、デフォルトテーブル
aField	フィールド	⇒	各レベルのソートするフィールド
> または <	演算子	⇒	各レベルのソート方向: >: 昇順, または <: 降順
*	演算子	⇒	ソート継続フラグ

説明

ORDER BYは、カレントプロセスのaTableのカレントレコードセレクションをソートします。ソートが終了すると、セレクションの先頭レコードがカレントレコードとなります。

aTable引数を省略した場合、コマンドはデフォルトテーブルに適用されます (デフォルトテーブルが事前に設定されていれば)。デフォルトテーブルが設定されていない場合、4Dは引数として渡された最初のフィールドのテーブルを使用します。引数を渡さず、デフォルトテーブルも設定されていない場合、エラーが生成されます。

aField、> または <、* 引数を指定しない場合、ORDER BYコマンドはaTableを対象とした並び替えエディタを表示します:



並び替えエディタの使用に関する詳細は、4D Design Referenceを参照してください。

ユーザはソートを組み立て、並び替えボタンをクリックしてソートを実行します。ソートが中断されずに実行されると、システム変数OKに1がセットされます。ユーザがキャンセルボタンをクリックすると、ORDER BYは中止されてソートは実行されず、システム変数OKには0がセットされます。

例題 1

以下の例は[Products]テーブルを対象とした並び替えエディタを表示します:

```
ORDER BY([Products])
```

例題 2

以下の例は、デフォルトテーブルを対象とした並び替えエディタを表示します (デフォルトテーブルが設定されていれば):

```
ORDER BY
```

aFieldや> または < を指定すると、標準の並び替えエディタは表示されず、ソートはプログラムで定義されることになります。1つのレベル、または複数のレベルを用いてセレクションのソートを実行することができます。ソートレベルごとに、aFieldにフィールドを、ソート方向として > または < を指定します。> は昇順を、< は降順を意味します。

例題 3

以下の例は[Products]テーブルのカレントセレクションをnameフィールドで昇順に並べ替えます:

```
ORDER BY([Products];[Products]Name;>)
```

例題 4

以下の例は[Products]テーブルのカレントセレクションをnameフィールドで降順に並べ替えます:

```
ORDER BY([Products];[Products]Name;<)
```

例題 5

以下の例は[Products]テーブルのカレントセレクションをtypeとpriceフィールドで、両レベルとも昇順に並べ替えます:

```
ORDER BY([Products];[Products]Type;>;[Products]Price;>)
```

例題 6

以下の例は[Products]テーブルのカレントセレクションをtypeとpriceフィールドで、両レベルとも降順に並べ替えます:

```
ORDER BY([Products];[Products]Type;<;[Products]Price;<)
```

例題 7

以下の例は[Products]テーブルのカレントセレクションをtypeの昇順およびpriceの降順で並べ替えます:

```
ORDER BY([Products];[Products]Type;>;[Products]Price;<)
```

例題 8

以下の例は[Products]テーブルのカレントセレクションをtypeの降順およびpriceの昇順で並べ替えます:

```
ORDER BY([Products];[Products]Type;<;[Products]Price;>)
```

ソート方向引数 > または < を省略すると、デフォルトで昇順が使用されます。

例題 9

以下の例は[Products]テーブルをnameフィールドで昇順にソートします:

```
ORDER BY([Products];[Products]Name)
```

1つのフィールドのみの指定 (1レベルのソート) で、そのフィールドにインデックスが設定されている場合、インデックスを使用してソートします。インデックスが設定されていなかったり、複数のフィールドを指定した場合には、(複合インデックスの場合を除き) ソート処理はシーケンシャルに行われます。フィールドは、並び替えが行われるテーブルに属しているか、aTableへの自動リレートまたはマニュアルリレートが設定された1テーブルに属しています。この場合、ソート処理は常にシーケンシャルに行われます。
ソートされるフィールドの組が複合インデックスである場合、ORDER BYはそのインデックスを使用します

例題 10

以下の例は、[Products]Nameフィールドにインデックスが設定されている場合は、インデックスソートを実行します:

```
ORDER BY([Products];[Products]Name;>)
```

例題 11

以下の例は、フィールドにインデックスが設定されていてもいなくても、シーケンシャルソートを実行します:

```
ORDER BY([Products];[Products]type;>;[Products]Price;>)
```

例題 12

以下の例は、リレートフィールドを使用してシーケンシャルソートを実行します:

```
ORDER BY([Invoices];[Companies]Name;>) ` InvoicesはCompaniesテーブルのnameフィールドを使用してソートされる
```

例題 13

データベースに[Contacts]LastName + [Contacts]FirstNameの複合インデックスが設定されている場合、以下の例題は2レベルのソートをインデックスを使用して行います:

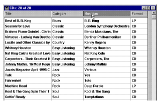
```
ORDER BY([Contacts];[Contacts]LastName;>;[Contacts]FirstName;>)
```

マルチソート (複数のフィールドによるソート) を実行するために、オプション引数 * を指定してORDER BYを何度も必要なだけ呼び出すことができます。実際のソートを始める最後のORDER BYにはこの引数を渡しません。この設定は、カスタマイズされたユーザインタフェースでの複数キーのソート管理に役立ちます。
警告: このシンタックスを利用した場合はORDER BYのコールごとに1つのソートレベル (フィールド) だけを渡すことができます。

例題 14

アプリケーションモードで表示される出力フォームで、ユーザが列ヘッダをクリックすると昇順にソートが行われるようにします。

Shiftキーを押しながら他の縦の列ヘッダをクリックすると、複数レベルでソートが実行されます:



Title	Category	Musician	Format
Best of B. B. King	Blues	B. B. King	LP
Season for Love	Classic	London Symphony Orchestra	CD
Brahms Piano Quintet - Clarin	Classic	Benda Musicians, The	CD
Virtuoso - Ludwig Van Beetho	Classic	Berliner Philharmoniker	CD
Lucille and Other Classics by	Country	Kenny Rogers	CD
Whitney Houston	Easy Listening	Whitney Houston	CD
Nat King Cole's Greatest Love	Easy Listening	Nat King Cole	CD
Carpenters - Their Greatest H	Easy Listening	Carpenters, The	CD
Johnny Mathis, 16 Most Requ	Easy Listening	Johnny Mathis	CD
Jazzis Magazine April 1995 C	Jazz	Various	CD
Talk	Rock	Yes	CD
Fahrenheit	Rock	Toto	CD
Machine Head	Rock	Deep Purple	LP
Kool & The Gang Spin Their T	Soul	Kool & The Gang	CD
Gettin' Ready	Soul	Temptations	CD

各列ヘッダには、以下のオブジェクトメソッドが記述されたハイライトボタンが置かれています:

```
MULTILEVEL(->[CDs]Title) `Title column header button
```

各ボタンは、列フィールドに対応するポインタを引数に、MULTILEVELプロジェクトメソッドを呼び出します。MULTILEVELプロジェクトメソッドは、以下の通りです:

```
  ` MULTILEVEL Project Method
  ` MULTILEVEL (Pointer)
  ` MULTILEVEL (->[テーブル]フィールド)

C_POINTER($1) `ソートレベル (フィールド)
C_LONGINT($lLevelNb)

  `ソートレベルを取得
If(Not(Shift down)) `1レベルのソート
   ARRAY POINTER(aPtrSortField;1)
   aPtrSortField{1}:=$1
Else
   $lLevelNb:=Find in array(aPtrSortField;$1) `このフィールドは既にソートされているか?
   If($lLevelNb<0) `されていなければ
      INSERT IN ARRAY(aPtrSortField;Size of array(aPtrSortField)+1;1)
      aPtrSortField{Size of array(aPtrSortField)}:=$1
   End if
End if
  `ソートを実行
$lLevelNb:=Size of(aPtrSortField)
If($lLevelNb>0) `最低1つのソートレベルがある
   For($i;1;$lLevelNb)
      ORDER BY([CDs];(aPtrSortField{$i})->;>;*) `ソート定義を構築
   End for
   ORDER BY([CDs]) `* を省略して、実際のソートを実行する
End if
```

ソートの定義方法に関係なく、実際のソート処理に時間がかかる場合は、4Dは自動的にサーモメータで進捗状況のメッセージを表示します。MESSAGES ONコマンドとMESSAGES OFFコマンドを使用して、メッセージを表示または非表示にすることができます。進捗サーモメータが表示された場合、ユーザは中止ボタンをクリックしてソートを中止することができます。ソートが正常に終了すると、システム変数OKには1がセットされ、それ以外の場合には0がセットされます。

ORDER BY FORMULA

ORDER BY FORMULA (aTable {; expression {; > or <}}{; expression2 ; > or <2 ; ... ; expressionN ; > or <N})

引数	型		説明
aTable	テーブル	⇒	セレクションをソートするテーブル
expression	式	⇒	各レベルのソートに設定する式 (文字, 実数, 整数, 倍長整数, 日付, 時間または ブール)
> or <	演算子	⇒	各レベルのソート方向: >: 昇順, または <: 降順

説明

ORDER BY FORMULAは、カレントプロセスのaTableのカレントレコードセレクションをソートします。ソートが終了すると、セレクションの先頭レコードがカレントレコードとなります。

引数aTableを必ず指定しなければならない点に注意してください。デフォルトテーブルを使用することはできません。

1つのレベルまたは複数のレベルを用いてセレクションのソートを実行できます。ソートレベルごとに、expressionと> または <を指定します。> は昇順を、< は降順を意味します。ソート方向を省略した場合、デフォルトとしてソートを昇順に行います。

引数expressionに指定できるタイプは、文字、実数、整数、倍長整数、日付、時間、ブールです。

ソートの定義方法に関係なく、実際のソート処理に時間がかかる場合は、4Dは自動的にサーモメータで進捗状況のメッセージを表示します。MESSAGES ONコマンドとMESSAGES OFFコマンドを使用して、メッセージを表示または非表示にすることができます。進捗サーモメータが表示された場合、ユーザは中止ボタンをクリックしてソートを中止することができます。ソートが正常に終了すると、システム変数OKには1がセットされ、それ以外の場合には0がセットされます。

4D Server: 4D Serverのバージョン11より、このコマンドはサーバ上で実行され、実行が最適化されるようになりました。expression内で直接変数が呼ばれているとき、クライアントマシンの変数値を使用してソートを計算します。
他方、フォーミュラにメソッドを使用し、メソッドから変数を参照する場合にはこの原則は適用されません (サーバ上の変数値を使用して評価が行われます)。このコンテキストでは、変数を引数として渡つつサーバ上でメソッドの時移行を可能にする、"サーバ上で実行"メソッド属性を使用することをお勧めします (Design Referenceマニュアルを参照)。
以前のバージョンの4D Serverでは、このコマンドはクライアントマシンで実行されていました。互換性のため、バージョン11に変換されたデータベースでは古い動作が保持されています。互換性の環境設定やSET DATABASE PARAMETER コマンドのセレクタを使用して、サーバ上で実行させるバージョン11の動作を適用できます。

例題

以下の例は、[People]テーブルをLastNameフィールドの文字長をキーにして降順に並び替えます。最も長い名字を持った人がカレントセレクションの先頭になります:

```
ORDER BY FORMULA([People];Length([People]LastName);<)
```

QUERY

QUERY ({aTable }{;}{ queryArgument {; *}})

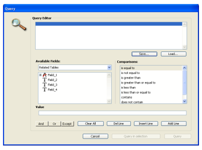
引数	型		説明
aTable	テーブル	→	レコードのセレクションを求めるテーブル, または 省略した場合、デフォルトテーブル
queryArgument	式	→	検索条件
*	演算子	→	検索継続フラグ

説明

QUERY は、aTableに対してqueryArgumentに指定した条件に一致するレコードを検索し、検索結果をセレクションとして返します。**QUERY**は、カレントプロセスのaTableのカレントセレクションを変更し、セレクションの先頭のレコードをカレントレコードにします。

aTable引数を省略した場合、コマンドはデフォルトテーブルに対して適用されます。デフォルトテーブルが設定されていない場合には、エラーが発生します。

queryArgumentまたは * を指定しない場合、**QUERY**はaTable 用のクエリエディタを表示します (複数クエリの最後の行である場合を除く、例題 2 参照):



クエリエディタについての詳細は4D Design Referenceマニュアルを参照してください。

ユーザはクエリを作成し、クエリボタンをクリックするか、絞り込みクエリボタンをクリックして検索を実行します。検索が中断されずに実行された場合、システム変数OKには1が代入されます。ユーザが「キャンセル」をクリックするか、**QUERY**コマンドが中断されて実際には検索が行われなかった場合には、システム変数OKに0が代入されます。

例題 1

以下の例は、[Products]テーブルに対するクエリエディタを表示します:

```
QUERY([Products])
```

例題 2

以下の例は、デフォルトテーブル (設定されている場合) に対するクエリエディタを表示します:

```
QUERY
```

queryArgument引数を指定すると、標準のクエリエディタは表示されず、クエリはプログラムから定義されます。単一クエリの場合 (1つのフィールドだけを検索)、queryArgumentをもとにQUERYコマンドを1回コールします。複合検索の場合 (複数フィールドに対する検索、または複合条件による検索)、queryArgumentを用いて必要な回数だけQUERYコマンドをコールします。そして、最後のQUERYコマンドのコール以外のQUERYコマンドに対してオプション引数 * を指定します。* を含まない最後のコールの後、実際の検索処理が実行されます。引数queryArgumentについては、この節で更に詳しく説明します。

例題 3

以下の例は、[People]テーブルの名前が"a"で始まる人のレコードを検索します:

```
QUERY([People];[People]Last name="a@")
```

例題 4

以下の例は、[People]テーブルの名字フィールドが"a"または"b"で始まる人のレコードを検索します:

```
QUERY([People];[People]Name="a@";*) // * は追加の検索条件があることを示します
QUERY([People];|;[People]Name="b@") // * を指定しないことでクエリ条件のスタックが終了したことを示します
```

注: @ 文字の解釈ルールは環境設定で変更できます。詳細は比較演算子を参照してください。

検索条件の指定

queryArgument 引数には以下のシンタックスを指定できます:

{ 論理演算子 ; } フィールド 比較演算子 値

論理演算子は**QUERY**呼び出しを結合するために使用します。使用できる論理演算子はクエリエディタで使用できるものと同じです:

論理演算子	QUERYで使用する記号
AND	&
OR	|
Except	#

論理演算子はオプションで、複合検索の最初の**QUERY**コマンドおよび**QUERY**コマンドが1つしかない場合には使用されません。複合検索クエリで論理演算子を省略した場合、デフォルトで**AND** (&) が使用されます。

fieldは検索対象となるフィールドです。aTableが1テーブルへの自動リレートまたはマニュアルリレートを持つ場合、fieldはリレート先の1テーブルのフィールドも使用可能です。

比較演算子はフィールドと値とを比較するために使用します。比較演算子を次に示します:

比較演算子	QUERYで使用する記号
等しい	=
等しくない	#
より少ない	<
より多い	>
以下	<=
以上	>=
キーワードを含む	%

Note: 比較演算子を記号ではなく文字式でも指定できます。この場合、クエリ文字列の項目を区切るためにセミコロンを使用しなければなりません。この方法を使用すれば、例えばカスタムクエリのユーザインタフェースを作成することができます。例題 20を参照してください。

値はフィールドと比較するためのデータです。値は、評価結果がフィールドと同じデータタイプである式です。値は検索の初めに一度だけ評価されます。各レコードに対して評価されるわけではありません。文字列中に特定の文字列を含んでいるかどうかを検索する場合 (包含検索) には、値にワイルドカード記号 (@) を使用することができます。

キーワードによる検索は文字やテキスト型のフィールドに対してのみ可能です。このタイプのクエリに関する詳細は比較演算子を参照してください。

複合条件検索を実行するための規則を次に示します:

- 最初の**QUERY**は論理演算子を含んではいけません。
- 最初以外の**QUERY**は論理演算子から始めることができます。論理演算子を省略した場合、デフォルトでAND (&) が使用されます。
- 最後の**QUERY**以外は引数 * を指定しなければいけません。
- 複合検索を実行するためには、最後の**QUERY**で引数 * を指定してはいけません。代りに最後の**QUERY**で、テーブル以外に引数のない**QUERY**コマンドを実行することができます (クエリエディタは表示されず、前の行までに定義した複合検索が実行されます)。

注: 作成したカレントクエリは各テーブル毎に管理されます。つまり各テーブルに1つの複合検索を同時に作成できます。aTable引数を指定するかデフォルトテーブルを指定することで、検索対象テーブルを明確に特定する必要があります。

検索の定義方法に関係なく、以下の処理が行われます:

- 実際の検索処理に時間がかかる場合、4Dは自動的にサーモメータで進捗状況のメッセージを表示します。MESSAGES ONコマンドとMESSAGES OFFコマンドを使用して、メッセージを表示または非表示にすることができます。進捗サーモメータが表示された場合、ユーザは中止ボタンをクリックして検索を中断することができます。検索が正常に終了すると、システム変数OKには1がセットされ、それ以外の場合には0がセットされます。
- インデックスフィールドが指定された場合、検索処理は毎回最適化され (最初にインデックスフィールドが検索される)、検索は最小の時間で終了します。コマンドは**AND** (&) を使用するクエリでは、複合インデックスを使用できます。

例題 5

以下の例は、Smithという名の人のレコードをすべて検索します:

```
QUERY([People];[People]Last Name="Smith")
```

Note: Last Nameのフィールドにインデックスが設定されている場合、QUERYコマンドは高速な検索のために、自動的にインデックスを使用します。

Reminder: この検索では、"Smith"、"smith"、"SMITH"等のレコードを検索します。大文字と小文字を区別するには、文字コードを使用した検索を行ってください。

例題 6

以下の例はJohn Smithという名前の人のレコードをすべて検索します。Last Nameのフィールドにはインデックスが設定されていますが、First Nameのフィールドにはインデックスが設定されていません。

```
QUERY([People];[People]Last Name="smith";*) ` Find every person named Smith
QUERY([People];&;[People]First Name="john") ` with John as first name
```

検索を実行すると、まず最初にインデックスを持ったLast Nameのフィールドを検索し、Smithという名前の人のレコードのみに検索対象を絞ります。次に、このレコードセレクションの中からFirst Nameのフィールドに対してシーケンシャルな検索を行います。

Note: このクエリはデータベースに[People]Last Name+[People]First Nameの複合インデックスが設定されていると特に最適化されます。この場合コマンドはインデックスを使用できます。

例題 7

以下の例はSmithまたはJonesという名字の人のレコードをすべて検索します。名字 (Last Name) のフィールドにはインデックスが設定されています。

```
QUERY([People];[People]Last Name="smith";*) ` Find every person named Smith…
QUERY([People];|;[People]Last Name="jones") ` ...or Jones
```

QUERYコマンドは両方の検索にLast Nameインデックスを使用します。2つの検索が実行され、その結果が内部セットに納められて、結合されます。

例題 8

以下の例は、会社名が設定されていないレコードをすべて検索します。これは空のフィールド(空の文字列)を検索することによって行います。

```
QUERY([People];[People]Company="") ` Find every person with no company
```

例題 9

以下の例は、Smithという名字でNew Yorkに会社のある人のレコードをすべて検索します。2番目のQUERYコマンドは、他のテーブルのフィールドを使用しています。これは、[People]テーブルから[Company]テーブルへN対1リレートされているために可能になっています:

```
QUERY([People];[People]Last Name="smith";*) ` Find every person named Smith…
QUERY([People];&;[Company]State="NY") ` ... who works for a company based in NY
```

例題 10

以下の例は、名前のイニシャルがAからMの人のレコードをすべて検索します:

```
QUERY([People];[People]Name<"n") ` Find every person from A to M
```

例題 11

以下の例は、郵便番号がが"94" (サンフランシスコ) または"90" (ロサンゼルス) の人のレコードをすべて検索します:

```
QUERY([People];[People]ZIP Code ="94@";*) ` Find every person in the SF…
QUERY([People];|;[People]ZIP Code ="90@") ` ...or Los Angeles areas
```

例題 12

キーワードによる検索: 以下の例題は[Products] テーブルのdescriptionフィールドに単語"easy"が含まれるレコードを検索します:

```
QUERY([Products];[Products]description%"easy")
  ` Find products whose 説明 contains the keyword easy
```

例題 13

以下の例題は、リクエストダイアログに入力したインボイス参照と等しい請求書を検索します:

```
vFind:=Request("Find invoice reference:") ` Get an invoice reference from the user
If(OK=1) ` If the user pressed OK
   QUERY([Invoice];[Invoice]Ref=vFind) ` Find the invoice reference that matches vFind
End if
```

例題 14

以下の例は、1996年に作成された請求書のレコードをすべて検索します。これは、1995年12月31日よりあとで1997年1月1日より前のレコードをすべて検索します:

```
QUERY([Invoice];[Invoice]In>!12/31/95!;*) ` Find invoices after 12/31/95…
QUERY([Invoice];&;[Invoice]In<!1/1/97!) ` and before 1/1/97
```

例題 15

以下の例は、給与が$10,000以上で、$50,000未満の条件に当てはまる従業員のレコードをすべて検索します:

```
QUERY([Employee];[Employee]Salary >=10000;*) ` Find employees who make between…
QUERY([Employee];&;[Employee]Salary <50000) ` ...$10,000 and $50,000
```

例題 16

以下の例は、$20,000以上の給与を得ているマーケティング部に所属する従業員のレコードをすべて検索します。Salaryフィールドにはインデックスが設定されているため最初に検索されます。2番目のQUERYコマンドが、他のテーブルのフィールドを使用している点に注意してください。[Employee]テーブルから[Dept]テーブルへn対1の自動リレートが設定されているためにこのようなことが可能になっています:

```
QUERY([Employee];[Employee]Salary >20000;*) ` Find employees with salaries over $20,000 and...
QUERY([Employee];&;[Dept]Name="marketing") ` ...who are in the marketing department
```

例題 17

N対1でリレートされた3つのテーブルがあります: [City] -> [Department] -> [Region]。以下のクエリは市の名前が"Saint"で始まるRegionを検索します:

```
QUERY([Region];[City]Name="Saint@") ` Find all the regions with cities beginning with "Saint"
```

例題 18

以下はmyVar変数の値を使用して情報を検索します。

```
QUERY([Laws];[Laws]Text =myVar) ` Find all laws that match myVar
```

myVarの値により、クエリは様々な結果になります。例えば:

- myVarが"Copyright@"のとき、Copyrightで始まる文字列を検索します。
- myVarが"@Copyright@"のとき、Copyright文字列を含むレコードを検索します。

例題 19

以下の例題では、変数の値に基づき複合クエリを組み立てます。この方法で、有効な条件のみを検索に使用することができます:

```
QUERY([Invoice];[Invoice]Paid=False;*)
If($city#"") ` if a city name has been specified
   QUERY([Invoice];[Invoice]Delivery_city=$city;*)
End if
If($zipcode#"") ` If a zip code has been specified
   QUERY([Invoice];[Invoice]ZipCode=$zipcode;*)
End if
QUERY([Invoice]) ` Execution of query on the criteria
```

例題 20

この例題では比較演算子に文字式を使用する方法を示します。比較演算子はポップアップメニューでカスタムクエリダイアログボックスに置かれています:

```
C_TEXT($oper)
$oper:=_popup_operator{_popup_operator} `$oper equals for example "#" or "="
If(OK=1)
   QUERY(Invoice];[Invoice]Amount;$oper;$amount)
End if
```

例題 21

ピクチャーフィールドのキーワードインデックスを使用すると、アプリケーションの速度を劇的に向上できます。

```
QUERY([PICTURES];[PICTURES]Photos %"cats") // "cats"キーワードが割り当てられた画像を検索する
```

システム変数およびセット

クエリが正しく実行されると、OKシステム変数が1に設定されます。以下の場合は0に設定されます:

- クエリダイアログボックスでユーザがキャンセルボタンをクリックした。
- "QUERY and LOCK"モード (SET QUERY AND LOCKコマンド参照)、クエリが一つ以上のロックされたレコードを見つけた。この場合、LockedSetシステムセットが更新されます。

QUERY BY EXAMPLE

QUERY BY EXAMPLE ({aTable}{;}{*})

引数	型		説明
aTable	テーブル	⇒	レコードのセレクションを返すテーブル, または 省略した場合、デフォルトテーブル
*	演算子	⇒	指定した場合、スクロールバーの非表示

説明

QUERY BY EXAMPLEは、デザインモードのフォームによるクエリメニューと同じ処理を実行します。このコマンドはクエリウィンドウとしてカレント入力フォームを表示します。QUERY BY EXAMPLEは、クエリウィンドウに入力された情報を使用してaTableを検索します。このときに使用するフォームには、ユーザに検索させたいフィールドを置かなければなりません。この検索は最適化されています。つまり、クエリを最適化するためにインデックスフィールドが自動的に使用されます。

デザインモードのフォームによるクエリメニューの使用方法に関する詳細は、4D Design Referenceマニュアルを参照してください。

例題

以下のメソッドの例は、まずMyQueryという名前のフォームを表示します。ユーザがフォームに検索データを入力し、検索の実行を要求した場合（システム変数OKに1が代入された場合）に、検索結果を表示するようにします:

```
FORM SET INPUT([People];"MyQuery") ` 検索フォームに切り替える
QUERY BY EXAMPLE([People]) ` フォームを表示し、クエリを実行
If(OK=1) ` ユーザがクエリを実行すると
   DISPLAY SELECTION([People]) ` レコードを表示
End if
```

システム変数およびセット

ユーザが入力ボタンをクリックするか、Enterキーを押すと、システム変数OKが1に設定され、クエリが実行されます。ユーザがキャンセルボタンをクリックするか、キャンセルのキーコンビネーションを押すと、OKシステム変数が0に設定され、クエリはキャンセルされます。

QUERY BY FORMULA

QUERY BY FORMULA (aTable {; queryFormula})

引数	型	説明
aTable	テーブル	⇒ レコードセレクションを求めるテーブル
queryFormula	ブール	⇒ 検索フォーミュラ

説明

QUERY BY FORMULAはaTableからレコードを検索します。**QUERY BY FORMULA**は、カレントプロセスのaTableのカレントセレクションを変更し、セレクションの先頭のレコードをカレントレコードにします。

QUERY BY FORMULAとQUERY SELECTION BY FORMULAは、全く同じように機能しますが、**QUERY BY FORMULA**がテーブルのすべてのレコードを検索対象とするのに対して、QUERY SELECTION BY FORMULAコマンドはカレントセレクションのレコードのみを検索対象とします。

両方のコマンドは、テーブルまたはセレクションの各レコードに対してqueryFormulaを適用します。queryFormulaはTRUEかFALSEのいずれかの状態に評価されるブール式です。queryFormulaでTRUEに評価されたレコードを新しいセレクションに追加します。

queryFormulaは、フィールドと値とを比較するだけの単純なものから、計算、またはリレート先テーブルの情報を評価するような複雑なものまで処理します。queryFormulaには4Dの関数 (コマンド) や開発者が作成した関数 (メソッド) や式 (フォーミュラ) を使用することができます。文字フィールドやテキストフィールドに対して作業を実行する場合は、queryFormulaにワイルドカード (@) を使用することもできます。詳しい情報はQUERYコマンドの例を参照してください。

queryFormulaが省略されると、4Dはフォーミュラでクエリダイアログボックスを表示します。

検索が完了すると、新しいセレクションの最初のレコードがディスクからロードされカレントレコードになります。

これらのコマンドは最適化され、特にインデックスを利用します。クエリのタイプが許す場合、これらのコマンドはQUERYコマンドと同じのクエリを実行します。例えば

QUERY BY FORMULA([mytable]; [mytable]myfield=value)

は可能であればインデックスを使用し、

QUERY([mytable]; [mytable]myfield=value)

と同じに実行されます。4Dは最適化可能な部分を先に検索し、他の残りのクエリと合算することで、部分的に最適化できないクエリも最適化します。例えば、

QUERY BY FORMULA([mytable];Length(myfield)=value)

は最適化されません。他方、

QUERY BY FORMULA([mytable];Length(myfield)=value1 | myfield=value2)

は部分的に最適化されます。

これらのコマンドはデフォルトでSQLのような"JOIN"を行います。これは

QUERY BY FORMULA([Table_A];[Table_A]field_X = [Table_B]field_Y)

のようなステートメントを実行する際にも、Table_AとTable_Bの間に自動リレーションが必要ないことを意味します (例題 3参照)。ストラクチャーエディター中でリレーションが存在しても、ルールとしては使用されません。しかし以下のケースでは、これらのコマンドは自動リレーションを使用します:

- フォーミュラが { フィールド ; 比較演算子; 値} の形式の要素に分解できない場合
- 同じテーブルの2つのフィールドが比較されている場合

Note: 互換性のためJOINメカニズムを無効にできます。これはデータベース環境設定でグローバルに (変換されたデータベースのみ)、またはSET DATABASE PARAMETERコマンドコマンドを使用してプロセスごとに行うことができます。

4D Server: 4D Serverのバージョン11より、このコマンドはサーバ上で実行され、実行が最適化されるようになりました。queryFormula内で直接変数が呼ばれているとき、クライアントマシンの変数値を使用してクエリを計算します。例えば**QUERY BY FORMULA**([mytable];[mytable]myfield=myvariable)というステートメントはサーバ上で実行されますが、myvariable変数の内容はクライアントマシンのものが使用されます。
他方、フォーミュラにメソッドを使用し、メソッドから変数を参照する場合にはこの原則は適用されません (サーバ上の変数値を使用して評価が行われます)。このコンテキストでは、変数を引数として渡つつサーバ上でメソッドの時移行を可能にする、"サーバ上で実行"メソッド属性を使用することをお勧めします (Design Referenceマニュアルを参照)。
以前のバージョンの4D Serverでは、このコマンドはクライアントマシンで実行されていました。互換性のため、バージョン11に変換されたデータベースでは古い動作が保持されています。互換性の環境設定やSET DATABASE PARAMETERコマンドのセレクタを使用して、サーバ上で実行させるバージョン11の動作を適用できます。

例題 1

以下の例は、すべての年の12月に作成された請求書のレコードを検索します。これはMonth of関数を各レコードに適用して検索します。このような検索は月の情報を別のフィールドとして持たない限り、他の方法では実現できません:

```
QUERY BY FORMULA([Invoice];Month of([Invoice]Entered)=12) ` Find the invoices entered in December
```

例題 2

以下の例は、名前が10文字を超える人のレコードを検索します:

```
QUERY BY FORMULA([People];Length([People]Name)>10) ` Find names longer than ten characters
```

例題 3

以下の例は特定のフォーミュラを使用したクエリに対しSQL JOINを有効にします:

```
$currentVal:=Get database parameter(QUERY BY FORMULA Joins)
SET DATABASE PARAMETER(QUERY BY FORMULA Joins;2) `Activate SQL joins
 `Query all the lines of "ACME" client invoices even though the tables are not related
QUERY BY FORMULA([invoice_line];([invoice_line]invoice_id=[invoice]id&[invoice]client="ACME"))
SET DATABASE PARAMETER(QUERY BY FORMULA Joins;$currentVal) `We re-establish the current settings
```

QUERY SELECTION

QUERY SELECTION ({aTable }{;}{ queryArgument {; *}})

引数	型		説明
aTable	テーブル	⇒	レコードセレクションを求めるテーブル, または 省略した場合、デフォルトテーブル
queryArgument	式	⇒	検索条件
*	演算子	⇒	検索継続フラグ

説明

QUERY SELECTIONは、aTableのレコードを検索します。QUERY SELECTIONはカレントプロセスのaTableのカレントセレクションを変更し、セレクションの先頭のレコードをカレントレコードにします。

QUERY SELECTIONは、QUERYと同じような動作を実行します。相違点は検索する範囲が異なるだけです:

- QUERYはテーブル中全レコードの中からレコードを検索します。
- QUERY SELECTIONはテーブルのカレントセレクションの中からレコードを検索します。

詳細については、QUERYコマンドの説明を参照してください。

例題

以下の例は、QUERY SELECTIONとQUERYの違いを表わしたものです。2種類の検索があります:

```
  ` New York Cityにあるすべての会社を検索する
QUERY([Company];[Company]City="New York City")
  ` Stock Exchange ビジネスを行っている会社を検索する
  ` 所在地は関係ない
QUERY([Company];[Company]Type Business="Stock Exchange")
```

2つめのQUERYは1番目の結果を無視していることに留意してください。以下では:

```
  ` New York Cityにあるすべての会社を検索する
QUERY([Company];[Company]City="New York City")
  ` New York Cityにある会社の中から
  ` Stock Exchange ビジネスを行っている会社を検索する
QUERY SELECTION([Company];[Company]Type Business="Stock Exchange")
```

QUERY SELECTIONは選択されたレコードを対象に検索を行います。

QUERY SELECTION BY FORMULA

QUERY SELECTION BY FORMULA (aTable {; queryFormula})

引数	型		説明
aTable	テーブル	⇒	レコードセレクションを求めるテーブル
queryFormula	ブール	⇒	クエリフォーミュラ

説明

QUERY SELECTION BY FORMULAは、aTableからレコードを検索します。QUERY SELECTION BY FORMULAは、カレントプロセスのaTableのカレントセレクションを変更し、セレクションの先頭のレコードをカレントレコードにします。

QUERY SELECTION BY FORMULAはQUERY BY FORMULAと同じような動作を実行します。相違点は検索する範囲が異なるだけです:

- QUERY BY FORMULAはテーブル中全レコードの中からレコードを検索します。
- QUERY SELECTION BY FORMULAはテーブルのカレントセレクションの中からレコードを検索します。

詳細については、QUERY BY FORMULAの説明を参照してください。

QUERY SELECTION WITH ARRAY

QUERY SELECTION WITH ARRAY (targetField ; array)

引数	型		説明
targetField	フィールド	→	値の比較に使用するフィールド
array	配列	→	検索する値の配列

説明

QUERY SELECTION WITH ARRAY コマンドは1番目の引数に渡されたフィールドのテーブルを検索し、targetField がarray要素のうち少なくともひとつの値に等しいレコードをカレントセレクションにします。

QUERY SELECTION WITH ARRAYはQUERY WITH ARRAYと同じような動作を実行します。相違点は検索する範囲が異なるだけです:

- QUERY WITH ARRAYは、targetFieldを含むテーブル中全レコードの中からレコードを検索します。
- QUERY SELECTION WITH ARRAYは、targetFieldを含むテーブルのカレントセレクションの中からレコードを検索します。

詳細は、QUERY WITH ARRAY コマンドの説明を参照してください。

QUERY WITH ARRAY

QUERY WITH ARRAY (targetField ; array)

引数	型		説明
targetField	フィールド	→	値との比較に使用するフィールド
array	配列	→	検索する値の配列

説明

コマンドは、targetFieldの値が少なくともarray要素のうち1つに等しいレコードを、targetFieldが属するテーブルの全レコードの中から検索します。検索されたレコードはカレントセレクションとなります。

このコマンドを使用すると、複数の値を使用する検索を素早く簡単に構築できます。

Notes:

- このコマンドはピクチャ、サブフィールド、BLOB 型のフィールドには使用できません。
- targetFieldとarrayは同じデータタイプでなければなりません。例外: 時間型のフィールドに対しては倍長整数配列を使用できます。

例題

以下の例題では、フランスとアメリカの顧客を検索します:

```
ARRAY STRING(2;SearchArray;2)
SearchArray{1}:="FR"
SearchArray{2}:="US"
QUERY WITH ARRAY([Clients]Country;SearchArray)
```

SET QUERY AND LOCK

SET QUERY AND LOCK (lock)
引数型　　説明
lock ブール→True = クエリで見つけたレコードをロック False = レコードをロックしない

説明

SET QUERY AND LOCK コマンドを使用して、カレントのトランザクション中でこのコマンドに引き続き呼び出されるすべてのクエリで見つかったレコードを自動的にロックできます。つまりクエリを行ってから結果を処理するまで、他のプロセスはレコードを変更できなくなることを意味します。

デフォルトで、検索されたレコードはロックされません。ロックを有効にするにはlock引数にTrueを渡します。

このコマンドはトランザクションの中で使用しなければなりません。このコマンドがトランザクションの外側で呼び出されると、エラーが生成されます。このコマンドはレコードロックのより良いコントロールを提供します。検索されたレコードはトランザクションが終了 (有効またはキャンセル) するまでロックされたままとなります。トランザクションが終了すると、レコードのロックは解除されます。

カレントトランザクション中のすべてのテーブルのレコードがロックされます。

SET QUERY AND LOCK(True) 文が実行されると、(QUERYのような) クエリコマンドは、すでにロックされたレコードを見つけると、特定の動作を選択します:

- クエリが停止され、システム変数OKは0に設定されます,
- カレントセレクションはクリアされます,
- LockedSetシステムセットにはクエリを停止する原因となったロックされたレコードが格納されます。

したがって、カレントセレクションが空だったりOK変数が0だった場合、LockedSetをテストして失敗の原因を検証する必要があります。

このメカニズムを無効にするには、SET QUERY AND LOCK(False) を実行します。

SET QUERY AND LOCK は、下記のクエリコマンドの動作を変更します:

- QUERY
- QUERY SELECTION
- QUERY BY EXAMPLE
- QUERY BY FORMULA
- QUERY BY SQL
- QUERY SELECTION BY FORMULA
- QUERY SELECTION WITH ARRAY
- QUERY WITH ARRAY

これに対してSET QUERY AND LOCKは、ALL RECORDSやRELATE MANY等、テーブルのカレントセレクションを変更する他のコマンドには影響を与えません。

例題

この例題では、CategoryがCに属する顧客は、QUERYとDELETE SELECTIONの間で、他のプロセスから変更や削除はできません。:

```
START TRANSACTION
SET QUERY AND LOCK(True)
QUERY([Customers];[Customers]Categorie=C)
 `At this moment, the records found are automatically locked for all other processes
DELETE SELECTION([Customers])
SET QUERY AND LOCK(False)
VALIDATE TRANSACTION
```

エラー管理

コマンドがトランザクションのコンテキスト中で呼び出されなかった場合、エラーが生成されます。

SET QUERY DESTINATION

SET QUERY DESTINATION (destinationType {; destinationObject {; destinationPtr}})

引数	型		説明
destinationType	倍長整数	⇨	0=カレントセレクション, 1=セット, 2=命名セレクション, 3=変数
destinationObject	文字, 変数	⇨	セット名、命名セレクション名、変数、または空の文字列
destinationPtr	ポインター	⇨	destinationType=3のとき変数へのポインター

説明

SET QUERY DESTINATIONを使い、カレントプロセスのクエリの結果を配置する場所を4Dに指示することができます。

destinationType引数に配置場所のタイプを指定します。Queriesテーマの定義済み定数を使用することもできます:

定数	型	値
Into current selection	倍長整数	0
Into named selection	倍長整数	2
Into set	倍長整数	1
Into variable	倍長整数	3

以下の表にしたがって、オプションのdestinationObject引数にクエリの配置先を指定します:

destinationType	destinationObject
引数	引数
0 (current selection)	引数を省略します
1 (set)	(既存のまたは作成させる) セット名
2 (named selection)	(既存のまたは作成させる) 命名セレクション
3 (variable)	(既存のまたは作成させる) 数値変数、またはdestinationPtr引数を使用する場合空の文字列 ("")

例:

```
SET QUERY DESTINATION(Into current selection)
```

以降の検索で見つかったレコードは、最終的にその検索の対象となるテーブルの新しいカレントセレクションとなります。

例:

```
SET QUERY DESTINATION(Into set;"mySet")
```

以降の検索で見つかったレコードは、最終的にセット"mySet"に配置されます。検索の対象となったテーブルのカレントセレクションとカレントレコードは変更されません。

注: クライアント/サーバでは、ローカル／クライアントセット ($で名前が始まるもの) をクエリの配置先として使用することはできません。このタイプのセットはサーバで検索が実行されたときにクライアントマシンで作られます。セットのタイプについての詳細はセットを参照してください。

例:

```
SET QUERY DESTINATION(Into named selection;"myNamedSel")
```

以降の検索で見つかったレコードは、最終的には命名セレクション"myNamedSel"に配置されます。検索の対象となったテーブルのカレントセレクションとカレントレコードは変更されません。

注:

- 命名セレクションが存在しない場合、検索が完了した時に自動的に作成されます。
- このコマンドはCUT NAMED SELECTIONと同様に命名セレクションを管理します。つまり参照だけが保持されます。命名セレクションをカレントセレクションに使用すると、命名セレクションは存在しなくなります。

例:

```
SET QUERY DESTINATION(Into variable;$vlResult)
```

または

```
SET QUERY DESTINATION(Into variable;"";->$vlResult)
```

以降の検索で見つかったレコード数が、変数$vlResultに配置されます。検索の対象となったテーブルのカレントセレクションとカレントレコードは変更されません。

警告: SET QUERY DESTINATIONは、カレントプロセス内で行われた以下の検索のすべてに影響を及ぼします。destinationTypeが0でない**SET QUERY DESTINATION**の呼び出しを行った後は、通常のクエリを再実行するために**SET QUERY DESTINATION**(0)の呼び出しと対にさせることを忘れないでください。

SET QUERY DESTINATIONは、下記のクエリコマンドの動作を変更します:

- QUERY
- QUERY SELECTION
- QUERY BY EXAMPLE
- QUERY BY FORMULA
- QUERY BY SQL
- QUERY SELECTION BY FORMULA
- QUERY SELECTION WITH ARRAY
- QUERY WITH ARRAY

これに対して**SET QUERY DESTINATION**はALL RECORDSやRELATE MANY等、テーブルのカレントセレクションを変更する他のコマンドには影響を与えません。

例題 1

以下の例は[Phone Book]テーブルのレコードを表示するフォームを作成します。そのフォームに (アルファベット26文字の) asRolodexと名付けられたタブコントロールと[Phone Book]レコードを表示するサブフォームを作成します。タブコントロールから任意のタブを選択することにより、そのタブ上の文字で始まるレコードを表示することができます。

このアプリケーション例では、[Phone Book] テーブルが変更されることはないため、任意のタブを選択するたびにクエリを実行する必要はありません。これにより、検索に要する時間を節約することができます。

これを実行するために、検索結果を命名セレクションに格納し、必要に応じて再利用できるようにします。以下のコードはasRolodexタブコントロールのオブジェクトメソッドです:

```
 ` asRolodex
Case of
   :(Form event=On Load)
 ` フォームが表示される前に
 ` rolodexと、検索が行われたかを示すブール配列を初期化
      ARRAY STRING(1;asRolodex;26)
      ARRAY BOOLEAN(abQueryDone;26)
      For($vlElem;1;26)
         asRolodex{$vlElem}:=Char(64+$vlElem)
         abQueryDone{$vlElem}:=False
      End for

   :(Form event=On Clicked)
 ` タブコントロールがクリックされたら、対応するクエリが実行済みかどうか調べる
      If(Not(abQueryDone{asRolodex}))
 ` 実行されていなければ、次のクエリ結果を命名セレクションに格納する
         SET QUERY DESTINATION(Into named selection;"temp")
 ` クエリを実行
         QUERY([Phone Book];[Phone Book]Last name=asRolodex{asRolodex}+"@")
 ` 通常のクエリモードに戻す
         SET QUERY DESTINATION(Into current selection)
 ` 検索されたレコードを使用する
         USE NAMED SELECTION("temp")
         COPY NAMED SELECTION([Phone book];"Rolodex"+asRolodex{asRolodex})
 ` 次回同じタブが選択された時は、クエリを行わない
         abQueryDone{asRolodex}:=True
      Else
 ` 選択された文字に対応するレコードを表示するために命名セレクションを使用
         USE NAMED SELECTION("Rolodex"+asRolodex{asRolodex})
      End if

   :(Form event=On Unload)
 ` フォームを閉じる際には
 ` 作成された命名セレクションをクリアする
      For($vlElem;1;26)
         If(abQueryDone{$vlElem})
            CLEAR NAMED SELECTION("Rolodex"+asRolodex{$vlElem})
         End if
      End for
 ` プロセス配列をクリアする
      CLEAR VARIABLE(asRolodex)
      CLEAR VARIABLE(abQueryDone)
End case
```

例題 2

この例題のUnique valuesプロジェクトメソッドは、テーブル中の任意の数のフィールドに対し、重複値がないことを検証するために使用できます。カレントレコードは新規あるいは既存のレコードを使用できます。

```
 ` Unique values プロジェクトメソッド
 ` Unique values ( ポインタ; ポインタ{ ; ポインタ... } ) -> ブール
 ` Unique values ( ->テーブル ; ->フィールド { ; ->フィールド2... } ) -> Yes or No

C_BOOLEAN($0)
C_POINTER(${1})
C_LONGINT($vlField;$vlNbFields;$vlFound;$vlCurrentRecord)
$vlNbFields:=Count parameters-1
$vlCurrentRecord:=Record number($1->)
If($vlNbFields>0)
   If($vlCurrentRecord#-1)
      If($vlCurrentRecord<0)
 ` カレントレコードはまだ保存されていない新規レコード (レコード番号= -3);
 ` なので、少なくとも1つのレコードが見つかれば検索を停止できる
         SET QUERY LIMIT(1)
      Else
 ` カレントレコードは既存のレコード;
 ` なので、少なくとも2つのレコードが見つかれば検索を停止できる
         SET QUERY LIMIT(2)
      End if
 ` クエリはカレントセレクションやカレントレコードを変更せずに
 ` $vlFoundに結果を返す
      SET QUERY DESTINATION(Into variable;$vlFound)
 ` 指定した数の¥フィールドのクエリを行う
      Case of
         :($vlNbFields=1)
            QUERY($1->;$2->=$2->)
         :($vlNbFields=2)
            QUERY($1->;$2->=$2->;*)
            QUERY($1->; & ;$3->=$3->)
         Else
            QUERY($1->;$2->=$2->;*)
            For($vlField;2;$vlNbFields-1)
               QUERY($1->; & ;${1+$vlField}->=${1+$vlField}->;*)
            End for
            QUERY($1->; & ;${1+$vlNbFields}->=${1+$vlNbFields}->)
      End case
      SET QUERY DESTINATION(Into current selection) ` クエリモードを通常に戻す
      SET QUERY LIMIT(0) ` クエリの上限をクリア
 ` クエリ結果を処理
      Case of
         :($vlFound=0)
            $0:=True ` 重複値はない
         :($vlFound=1)
            If($vlCurrentRecord<0)
               $0:=False ` 未保存の新規レコードと同じ値を持つ既存のレコードを見つけた
            Else
               $0:=True ` 重複値は見つからなかった
            End if
         :($vlFound=2)
            $0:=False ` ケースに関わらず、値は重複している
      End case
   Else
      If(<>DebugOn) ` Does not make sense; signal it if development version
         TRACE ` WARNING! カレントレコードなしでメソッドが呼び出された
      End if
      $0:=False ` 結果は保証できない
   End if
Else
   If(<>DebugOn) ` Does not make sense; signal it if development version
      TRACE ` WARNING! 検索条件なしでメソッドが呼び出された
   End if
   $0:=False ` 結果は保証できない
End if
```

このプロジェクトメソッドをアプリケーションから使用するには、以下のように記述します:

```
 ` ...
If(Unique values(->[Contacts];->[Contacts]Company);->[Contacts]Last name;->[Contacts]First name)
 ` Do appropriate actions for that record which has unique values
Else
   ALERT("There is already a Contact with this name for this Company.")
End if
 ` ...
```

SET QUERY LIMIT

SET QUERY LIMIT (limit)
引数型　　　説明
limit 倍長整数 ⇒ レコード数, または 0: 制限なし

説明

SET QUERY LIMITは、カレントプロセスの以降の検索を対象に、limitに渡した数のレコードが見つかったら検索を中止するよう4Dに指示します。

例えばlimitに1を渡すと、以降の検索は検索条件に一致した1件のレコードを見つけるとすぐにインデックスまたはデータファイルのブラウズ作業を中止します。

制限なしのクエリを再実行するには、limitに0を渡したSET QUERY LIMITを再度呼び出します。

警告: SET QUERY LIMITコマンドは、カレントプロセス内で行われる以降のクエリのすべてに影響を及ぼします。そのため、常にSET QUERY LIMIT(limit) (limit>0)の呼び出しは、制限なしのクエリを再実行するためのSET QUERY LIMIT(0)の呼び出しと対とすることを忘れないでください。

SET QUERY LIMITは、下記のクエリコマンドの動作を変更します:

- QUERY
- QUERY SELECTION
- QUERY BY EXAMPLE
- QUERY BY FORMULA
- QUERY SELECTION BY FORMULA
- QUERY BY SQL
- QUERY WITH ARRAY
- QUERY SELECTION WITH ARRAY

これに対して、SET QUERY LIMITは、ALL RECORDSやRELATE MANY等、テーブルのカレントセレクションを変更する他のコマンドには影響を与えません。

例題 1

"100万ドル以上の売上を獲得している顧客10人を探せ"という要求に対応する検索を実行するには、以下のように記述します:

```
SET QUERY LIMIT(10)
QUERY([Customers];[Customers]Gross sales>1000000)
SET QUERY LIMIT(0)
```

例題 2

SET QUERY DESTINATIONの2番目の例題参照

グラフ

- GRAPH Updated 14.0
- GRAPH SETTINGS Updated 14.0
- _o_GRAPH TABLE

GRAPH

GRAPH (graphArea ; graphNumber ; xLabels {; yElements} {; yElements2 ; ... ; yElementsN})

引数	型		説明
graphArea	ピクチャー変数	⇨	グラフエリアまたはピクチャ変数
graphNumber	倍長整数	⇨	グラフタイプ番号
xLabels	配列	⇨	X軸ラベル
yElements	配列	⇨	グラフにするデータ (最大8個)

説明

互換性に関する注意: *4D v14以降、GRAPH コマンドは第一引数にはピクチャ変数のみ受け取れるようになりました。グラフエリア(4D Chart)を使用するシンタックスは廃止され、サポートされていません。*

GRAPHコマンドは、フォームのグラフエリアやピクチャ変数に、配列のデータを使用してグラフを作成します。**GRAPH**コマンドはフォームメソッドまたはフォームに属するオブジェクトメソッド、あるいはこれら二つのメソッドから呼ばれるプロジェクトメソッドに置かなければなりません。

このコマンドで生成されるグラフは、統合されたSVGレンダリングエンジンを使用して描画されます。グラフにはピクチャー変数に関連付けられたイン ターフェース機能があります。(表示フォーマットの設定などに使用できる) アプリケーションモードのコンテキストメニュー、スクロールバー等。生成されたグラフには自動で特別なIDが割り当てられ、それらを個々に処理するために 使用できます (後述参照)。

注: SVG (Scalable Vector Graphics) はグラフィックファイルフォーマット (.svg 拡張子) です。XMLに基づき、このフォーマットは広く使用され、特にWebブラウザで表示できます。詳細は以下のWebサイトを参照してください: http://www.w3.org/Graphics/SVG/。SVG EXPORT TO PICTUREコマンドを使用して、統合されたSVGエンジンを利用することもできます。

graphPicture 引数にはフォーム中でグラフを表示するピクチャー変数名を渡します。

引数graphNumは描画されるグラフタイプを定義します。これは1から8までの数値でなければなりません。グラフタイプについては例題1を参照してください。グラフタイプを変更する場合は、グラフを作成した後でgraphNumを変更し、**GRAPH**コマンドをもう一度実行します。

引数 xLabels は、X軸（グラフの一番下）に使用するラベルを定義します。このデータは文字列、日付、時間、数値タイプのいずれでも構いません。xLabels とyElementsの配列要素数はそれぞれ同じでなければなりません。

引数yElementsで指定するデータは、グラフにするデータです。このデータは数値でなければなりません。最大8つのデータセットをグラフ化することができます。円グラフは、最初のyElementsのみをグラフ化します。

Automatic IDs

SVGグラフ中で見つけられる要素には自動で特定のIDが割り当てられるようになりました:

ID	説明
ID_graph_1 ~ ID_graph_8	棒、折れ線、面...
ID_graph_shadow_1 ~ ID_graph_shadow_8	グラフの影...
ID_bullet_1 ~ ID_bullet_8	点 (折れ線および散布図のみ)
ID_pie_label_1 ~ ID_pie_label_8	円グラフのラベル (円グラフのみ)
ID_legend_1 ~ ID_legend_8	凡例のタイトル
ID_legend_border	凡例の境界線
ID_legend_border_shadow	凡例境界線の影
ID_x_values	X軸の値
ID_y_values	Y軸の値
ID_y0_axis	Z軸の値
ID_background	背景
ID_background_shadow	背景の影
ID_x_grid	X軸のグリッド
ID_x_grid_shadow	X軸のグリッドの影
ID_y_grid	Y軸のグリッド
ID_y_grid_shadow	Y軸のグリッドの影

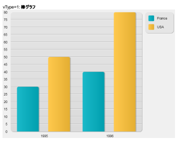
例題

以下の例は、グラフィックエンジンを使用して得ることのできる異なるグラフタイプを示します。コードはフォームメソッドあるいはオブジェクトメソッドに記述されます:

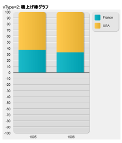
```
C_PICTURE(vGraph) // SVG エンジンを使用する場合に指定する
ARRAY TEXT(X;2) // X軸の配列を作成
X{1}:="1995" // X ラベル#1
X{2}:="1996" // X ラベル#2
ARRAY REAL(A;2) // Y軸の配列を作成
A{1}:=30 // データ挿入
A{2}:=40
ARRAY REAL(B;2) // Y軸の配列を作成
B{1}:=50 // データ挿入
B{2}:=80
vType:=1 // グラフタイプを初期化
GRAPH(vGraph;vType;X;A;B) // グラフ描画
GRAPH SETTINGS(vGraph;0;0;0;0;False;False;True;"France";"USA") // グラフの凡例をセット
```

以下の図はレンダリングエンジンごとのグラフの結果を示します。

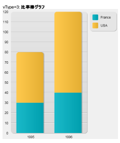
- vType=1: **棒グラフ**



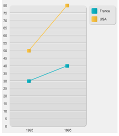
- vType=2: **積上げ棒グラフ**



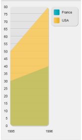
- vType=3: **比率棒グラフ**



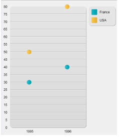
- vType=4: **線グラフ**



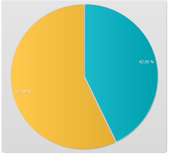
- vType=5: **面グラフ**



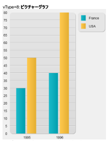
- vType=6: **点グラフ**



- vType=7: **円グラフ**



- vType=8: **ピクチャーグラフ**



注: ピクチャーはデフォルトでシンプルな四角です。

GRAPH SETTINGS

GRAPH SETTINGS (graph ; xmin ; xmax ; ymin ; ymax ; xprop ; xgrid ; ygrid ; title {; title2 ; ... ; titleN})

引数	型		説明
graph	ピクチャー変数	→	グラフエリアまたはピクチャ変数
xmin	倍長整数, 日付, 時間	→	比例グラフのx軸の最小値 (線または点グラフのプロットのみ)
xmax	倍長整数, 日付, 時間	→	比例グラフのx軸の最大値 (線または点グラフのプロットのみ)
ymin	倍長整数	→	y軸の最小値
ymax	倍長整数	→	y軸の最大値
xprop	ブール	→	TRUE: プロポーショナルX軸; FALSE: 通常のX軸 (線または点グラフのプロットのみ)
xgrid	ブール	→	TRUE: X軸グリッド; FALSE: X軸グリッドなし (xprop=TRUEの場合のみ)
ygrid	ブール	→	TRUE: Y軸グリッド; FALSE:Y軸グリッドなし
title	文字	→	凡例

説明

GRAPH SETTINGS はフォームに表示されるグラフの設定を変更します。グラフは GRAPH コマンドで表示済みでなければなりません。GRAPH SETTINGS は円グラフには効果ありません。このコマンドはフォームメソッドまたはフォームに属するオブジェクトメソッド、あるいはこれら二つのメソッドから呼ばれるプロジェクトメソッドに置かなければなりません。

xmin, xmax, ymin, そして ymax 引数はすべてそれぞれの軸の最小値と最大値を設定します。これらの引数ペアのいずれかがヌル値の場合 (0, ?00:00:00?, または !00/00/00!)、デフォルトのグラフ値が使用されます。

引数xpropは、線グラフ（タイプ4）と点グラフ（タイプ6）に対する比例プロットを有効にします。Trueであれば、点の値に従ってX軸上の各点をプロットします。ただし、値が数値、時間、日付の場合に限ります。

xgridとygridはグリッドラインを表示または非表示にします。X軸のグリッドは、プロットが比例する点グラフまたは線グラフの場合にのみ表示されます。

引数titleは、指定した文字列を凡例のラベルとして表示します。

例題

GRAPH コマンドの例題を参照。

_o_GRAPH TABLE

_o_GRAPH TABLE
このコマンドは引数を必要としません

説明

コマンドは削除されました*: このコマンドは4D v14以降サポートされなくなりました。*

コンパイラ

- コンパイラコマンド
- コンパイラ指示子の使用
- 型指定ガイド
- シンタックスの詳細
- 最適化のヒント
- エラーメッセージ
- C_BLOB
- C_BOOLEAN
- C_DATE
- C_GRAPH
- C_LONGINT
- C_OBJECT New 14.0
- C_PICTURE
- C_POINTER
- C_REAL
- C_TEXT
- C_TIME
- IDLE
- C_INTEGER
- C_STRING

コンパイラコマンド

統合された4Dのコンパイラは、データベースアプリケーションを機械語レベルに翻訳します。コンパイラの利点は次の通りです:

- **速度**: データベースの実行速度を3倍から1000倍速くします。
- **コードの検証**: データベースアプリケーションコードの整合性をチェックし、論理的矛盾や構文的矛盾を検出します。
- **保護**: データベースをコンパイルすると、インタプリタコードを削除できます。コンパイルされたデータベースは、故意的にも不注意からもストラクチャやメソッドの表示や修正することができないこと以外は、オリジナルのデータベースと同じに動作します。
- **スタンドアロンのダブルクリックで起動するアプリケーション**: コンパイル後のデータベースは、独自のアイコンを持つスタンドアロンアプリケーション(.EXEファイル)に作り変えることもできます。

コンパイラの操作については、Design Referenceマニュアルを参照してください。

このテーマのコマンドは、コンパイラの使用に関連があります。これらのコマンドは、データベース中のデータタイプを定義します。IDLEコマンドは、コンパイルされたデータベースで特別に使用されるコマンドです。

C_BLOB
C_BOOLEAN
C_DATE
C_GRAPH
C_OBJECT
C_INTEGER
C_LONGINT
C_PICTURE
C_POINTER
IDLE
C_REAL
C_STRING
C_TEXT
C_TIME

IDLEコマンド以外のこれらのコマンドは、変数を宣言し、それらを指定したデータタイプとしてキャストします。変数を宣言することによって、変数のデータタイプに関連する曖昧さが解決されます。変数がこれらのコマンドのいずれかで宣言されていない場合には、コンパイラが変数のデータタイプを判断しようとします。フォームで使用される変数のデータタイプは、多くの場合、コンパイラで判断するのは困難です。このため、開発者がこれらのコマンドによってフォームで使用される変数を宣言することが特に重要です。

Note: 時間を節約するために、コンパイラウインドウにあるオプションを使用し、(コンパイラメソッドと呼ばれる) 変数定義メソッドの生成や更新を行うことができます。このオプションは、データベース内で使用されるすべての変数のタイプ指定を行う変数定義メソッドを自動作成します。

配列は変数で、コンパイルについては標準変数と同じ規則に従わなければなりません。配列の宣言命令は配列テーマ内にまとめられています。

コンパイルされるコードの一般的な記述ルール

- 同じ名前を2つ以上のメソッドまたは変数に付けてはなりません。また、メソッドに変数と同じ名前をつけることはできません。
- バージョン1の4Dで使用されていた変数の間接参照はできません。パーセント記号(%)を使って間接的に変数を参照する文字型間接参照も、中カッコ({...})を用いる数値型間接参照も実行することはできません。中カッコ({...})は定義された配列の要素を指定する場合にのみ使用されます。しかし、引数に対する間接参照は使用できます。
- 変数や配列のデータタイプは変更できません。
- 1次元配列を2次元配列に、また2次元配列を1次元配列に変更することはできません。
- 文字(列)変数の長さや、文字列配列の要素の長さは変えられません。
- コンパイラにより変数のタイプ定義は行われますが、フォーム上の変数のように、データタイプが明確でない場合は、コンパイラ命令を使用して変数のデータタイプを指定するべきです。
- 変数を明示的にタイプ宣言するもう1つの理由は、コードの最適化です。このことは、特にカウンタとして使用される変数に対して当てはまります。最大限の能力を得るために、可能な限り倍長整数型の変数を使用してください。
- 変数をクリアする(各変数の初期値に設定する) には、変数名を用いてCLEAR VARIABLEコマンドを使用します。CLEAR VARIABLEコマンド内で変数の名前を表わす文字列を使用してはいけません。
- Undefined関数は、常にFalseを返します。変数は常に定義されています。
- 通常、倍長整数変数および整数変数の数値演算は、デフォルトの数値タイプ(実数)の演算よりもはるかに高速に実行されます。

これらの規則については、下記の節を参照してください:

- コンパイラ指示子の使用は、いつ、どこでコンパイラ命令を記述するかについて説明しています。
- 型指定ガイドは、4Dデータベースをコンパイルする際に起こりうるコンフリクトについて説明しています。
- ユーザページは、いくつかの4Dコマンドに関する追加情報を提供します。
- 最適化のヒントは、コンパイルモードで起動したアプリケーションをスピードアップするためのヒントを提供します。

例題 1

以下の例は、コンパイラ用の基本的な変数宣言です:

```
C_BLOB(vxMyBlob) ` プロセス変数vxMyBlobはBLOB型の変数として宣言されます
C_BOOLEAN(<>OnWindows) ` インタープロセス変数<>OnWindowsはブール型の変数として宣言されます
C_DATE($vdCurDate) ` ローカル変数$vdCurDateは日付型の変数として宣言されます。
C_GRAPH(vg1;vg2;vg3) ` 3つのプロセス変数vg1, vg2 そして vg3はグラフ型として宣言されます。
```

例題 2

以下の例は、プロジェクトメソッドOneMethodAmongOthersにて、3つの引数を宣言しています:

```
  ` OneMethodAmongOthers プロジェクトメソッド
  ` OneMethodAmongOthers ( Real ; Integer { ; Long } )
  ` OneMethodAmongOthers ( Amount ; Percentage { ; Ratio } )

C_REAL($1) ` 1番目の引数は実数
C_INTEGER($2) ` 2番目の引数は整数
C_LONGINT($3) ` 3番目の引数は倍長整数

  ` ...
```

例題 3

以下のプロジェクトメソッドCapitalizeは、文字列の引数を受けとり、文字列の結果を返します:

```
  ` Capitalize プロジェクトメソッド
  ` Capitalize ( String ) -> String
  ` Capitalize ( Source string ) -> Capitalized string

C_STRING(255;$0;$1)
$0:=Uppercase(Substring($1;1;1))+Lowercase(Substring($1;2))
```

例題 4

以下のプロジェクトメソッドSEND PACKETSは、引数として時間と、可変の数のテキスト引数を受け取ります:

```
  ` SEND PACKETS プロジェクトメソッド
  ` SEND PACKETS ( Time ; Text { ; Text2... ; TextN } )
  ` SEND PACKETS ( docRef ; Data { ; Data2... ; DataN } )

C_TIME($1)
C_TEXT(${2})
C_LONGINT($vlPacket)

For($vlPacket;2;Count parameters)
   SEND PACKET($1;${$vlPacket})
End for
```

例題 5

プロジェクトメソッドCOMPILER_Param_Predeclare28は、コンパイル用に他のプロジェクトメソッドのシンタックスを事前に宣言しています:

```
  ` COMPILER_Param_Predeclare28 プロジェクトメソッド

C_REAL(OneMethodAmongOthers;$1) ` OneMethodAmongOthers ( Real ; Integer { ; Long } )
C_INTEGER(OneMethodAmongOthers;$2) ` ...
C_LONGINT(OneMethodAmongOthers;$3) ` ...
C_STRING(Capitalize;255;$0;$1) ` Capitalize ( String ) -> String
C_TIME(SEND PACKETS;$1) ` SEND PACKETS ( Time ; Text { ; Text2... ; TextN } )
C_TEXT(SEND PACKETS;${2}) ` ...
```

コンパイラ指示子の使用

変数のデータ型

4Dの変数には次の3種類があります:

- ローカル変数
- プロセス変数
- インタープロセス変数

変数の詳細な性質についてはを参照してください。プロセス変数とインタープロセス変数はコンパイラにとって構造的に同じです。

プロセスにどの変数が使用されるか、コンパイラは知ることができないので、プロセス変数の使用にはインタープロセス変数よりも注意が必要です。新規プロセスの開始時にすべてのプロセス変数は複製されます。プロセス変数はプロセスごとに個別の値を持つことができますが、型はデータベース全体を通して同じです。

変数の型

すべての変数には型があります。データタイプ の節に記述されているとおり、変数には以下13の異なる型があります:

ブール
固定長文字列
日付
整数
倍長整数
グラフ
時間
ピクチャ
数値 (または実数)
ポインタ
テキスト
BLOB
オブジェクト

配列には12の型があります:

ブール配列
文字列配列
日付配列
整数配列
倍長整数配列
ピクチャ配列
実数配列
時間配列
オブジェクト配列
ポインタ配列
BLOB配列
テキスト配列

注:
- 4D v14以降、"オブジェクト"という型が使用できるようになりました。
- 4D v11より、文字タイプの変数とテキストタイプの変数に違いはなくなりました。今後はテキストタイプの利用が推奨されます。

シンボルテーブルの作成

インタプリタモードでは、変数は2つ以上のデータタイプを持つことが可能です。これは、コードをコンパイルするのではなく解釈する ためです。4Dはそれぞれのステートメントを別々に解釈し、そのコンテキストを理解します。しかしコンパイルモードの場合は状況が異なります。インタプリタが一行ずつ実行処理するのに対して、コンパイルの際はデータベース全体を処理します。

コンパイラの処理方法は以下のとおりです:

- コンパイラはデータベース内のオブジェクトをシステマチックに解析します。ここでいうオブジェクトとはデータベース、プロジェクト、フォーム、トリガ、オブジェクトメソッドです。
- コンパイラはオブジェクトを調べ、データベース内で使われる各変数のデータ型を決定し、変数やメソッドのテーブル（シンボルテーブル）を生成します。
- すべての変数のデータ型が確定すると、コンパイラはデータベースの翻訳（コンパイル）を行います。ただし、各変数のデータ型が確定できなければデータベースをコンパイルすることはできません。

同 じ変数名で異なる2つのデータ型が見つかった場合、どちらか1つが優先して採用されることはありません。オブジェクトの型を設定してメモリアドレスを割り 当てるために、コンパイラはオブジェクトについての名前やデータ型などを正確に認識する必要があります。コンパイラはデータ型からそのサイズを判断しま す。コンパイルされたデータベースごとにマップが作られ、各変数の名前（もしくは識別子）、場所（もしくはメモリアドレス）、変数が占める容量（データタ イプによってわかるもの）を記録します。このマップをシンボルテーブルと呼びます。環境設定のオプションでコンパイル時にこのテーブルをファイル形式で作 成するかどうかを選ぶことができます。
このマップは、コンパイラメソッドの自動生成にも使用されます。

コンパイラで型定義される変数

コンパイルで変数型を調べたりまた型定義を行ったりする場合は、簡単にコンパイラコマンドの記述場所を決めることができます。作業方法に応じて以下のいずれかの方法を選んでください:

- ローカル変数、プロセス変数、インタープロセス変数に応じて、変数が初めて使用されるメソッドやオブジェクトメソッドでコンパイラ指示子を使用します。コ ンパイラ指示子は、必ず変数が初めて使われる場所、つまり一番初めに実行されるはずのメソッドで使用するようにしてください。コンパイル時、コンパイラは 4Dで作成した順序でメソッドを処理します。エクスプローラで表示される順番ではない点に注意してください。
- コンパイラ指示子で定義するプロセス変数とインタープロセス変数は、またはから呼び出されるメソッドにまとめます。

ローカル変数については、使用するメソッドの一番初めにコンパイラ指示子をまとめてください。

いつコンパイラ指示子を使用するか

コンパイラ指示子は以下の場合、有用です:

- コンパイラで前後関係から変数のデータ型を決定できない場合
- 使用目的から型を決定させたくない場合。

またコンパイラコマンドを使用するとコンパイル時間を短縮できます。

曖昧な場合

コンパイラで変数型を決定できないこともあります。このような場合、コンパイラからはエラーメッセージが出力されます。
コンパイラでデータ型を決定できない場合は、主に3つの原因があります。データ型が複数ある場合、コンパイラの推測した型があいまいな場合、そして型を判断する情報がない場合です。

データ型が複数の場合
データベース中、異なるステートメントで変数の型が変更されていると、コンパイラはエラーを生成します。
コンパイラは最初に見つけた変数を選択し、そのデータ型を同じ名前の変数の次のオカレンスに適用します。しかしその変数の型が異なるとエラーになります。

例題:

メソッドA,

```
Variable:=True
```

メソッドB,

```
Variable:="The moon is green"
```

メソッドAがメソッドBよりも先にコンパイルされると、コンパイラはステートメント**Variable:="The moon is green"**でデータ型が変更されていると判断します。コンパイラは型の変更が行われていることを通知し、エラーを生成します。ほとんどの場合、2番目の変数のオカレンスの名前を変更することで、問題を解決できます。

コンパイラが決定した型が曖昧な場合
コンパイラが、オブジェクトの型が正しい方ではないと推定することがあります。この場合、コンパイラ指示子を使用して、変数の型を明示的に指定する必要があります。

以下はアクティブオブジェクト用のデフォルト値を使用した例です。
フォーム内で、プロパティリストのデータソーステーマ内にあるデフォルト 値用の編集ボタンを使用して、コンボボックス、ポップアップメニュー、タブコントロール、ドロップダウンリスト、メニュー／ドロップダウンリスト、および スクロールエリアのデフォルト値を割り当てることができます（詳細は4D Design Referenceを参照）。デフォルト値は、オブジェクト名と同じ名前の配列に自動でロードされます。
オブジェクトをメソッド内で使用しない場合、コンパイラはその配列を曖昧さなくテキスト配列と推測できます。
しかし初期化を行わなければならない場合、次のようにコーディングされることがあります:

```
Case of
   :(Form event=On Load)
      MyPopUp:=2
      ...
End case
```

この場合、曖昧さが出現します。メソッドを解析する際、コンパイラはオブジェクトMyPopUpを実数型と推定します。この場合、フォームメソッドまたはコンパイラメソッドで、配列を明示的に宣言する必要があります:

```
Case of
   :(Form event=On Load)
      ARRAY TEXT(MyPopUp;2)
      MyPopUp:=2
      ...
End case
```

型を判断する決め手がない場合
宣言なしに変数が使用されていて、前後関係からデータ型を決定できないような状況です。この場合、コンパイラにとっての決め手はコンパイラ指示子しかありません。
こうした状況は、主に次の4つのコンテキストにおいて起こります:

- ポインタを使用する場合
- 複数のシンタックスを持つコマンドを使用する場合
- 異なるデータ型のオプション引数を受け入れるコマンドを使用する場合
- URL経由で呼び出された4Dメソッドを使用する場合

ポインタ
ポインタは、自分自身以外のデータ型を返すことができません。
次のような場合:

```
Var1:=5.2(1)
Pointer:=->Var1(2)
Var2:=Pointer->(3)
```

(2) でPointerによりポイントされている変数の型が定義されているにもかかわらず、Var2の型を知ることはできません。コンパイル中、コンパイラはポ インタを認識できますが、ポインタが指す変数の型を知る方法はありません。結果、Var2に型を推定できません。このような場合、コンパイラ指示子が必要 です: C_REAL(Var2).

複数シンタックスコマンド
Year of関数に割り当てられた変数を使用する際、この関数の動作に基づき、変数の型は日付しかありえません。しかしことは常にそのように単純とは限りません。たとえば:
GET FIELD PROPERTIES コマンドは2つのシンタックスを受け入れます:
GET FIELD PROPERTIES(tableNo;fieldNo;type;length;index)
GET FIELD PROPERTIES(fieldPointer;type;length;index)

複数シンタックスコマンドを使用する際、コンパイラは選択されたシンタックスや引数を推測できません。データベースのどこかでその使用に基づく型付けが行われていない限り、コンパイラ指示子を使用して変数を型付けしなければなりません。

異なるデータ型のオプション引数を受け入れるコマンド
異なるデータ型の複数のオプション引数を受け入れるコマンドを使用する場合、コンパイラはどのオプション引数が使用されたか知ることができません。
例えば:
GET LIST ITEM コマンドは2つのオプション引数を受け入れます; 1番目は倍長整数で2番目はブールです。
コマンドは以下のいずれかの方法で使用されます:
GET LIST ITEM(list;itemPos;itemRef;itemText;sublist;expanded)
または:
GET LIST ITEM(list;itemPos;itemRef;itemText;expanded)
データベースのどこかでその使用に基づく型付けが行われていない限り、コンパイラ指示子を使用して変数を型付けしなければなりません。

URLで呼ばれるメソッド
URLから呼ばれる必要のある4Dメソッドを書くときで、メソッドで$1を使用しないときでも、以下のように明示的に$1をテキスト変数として宣言しなければなりません:
C_TEXT($1)
実際コンパイラは4DがURLから呼ばれるかどうか判断できません。

コンパイルの時間の短縮

データベースで使用する変数がすべて明示的に定義されていれば、コンパイラで型定義を調べる必要はありません。この場合、オプションを設定してメソッドの翻訳フェーズだけを行うように指定できます。この操作により、コンパイル時間を半分以下に短縮できます。

コードの最適化

コンパイラコマンドを使用することで、メソッドの処理速度を上げることができます。この件についての詳細はの 節を参照してください。簡単な例として、カウンタとして使用するローカル変数をインクリメントする必要があるとします。倍長整数として宣言すると、コンパイルしたデータベースの効率が良くなります。例えばWindowsの場合、実数データがメモリを8バイト使用するのに対し、倍長整数にすると、4バイトし か使用しません。8バイトのカウンタをインクリメントする場合は当然、4バイトの場合より時間がかかります。

コンパイラ指示子をどこに記述するか

コンパイラ指示子は、コンパイラに変数の型付けをまかせるかどうかによって以下2通りに処理方法が異なります。

変数の型設定

コンパイラは変数の判別基準を守り、以下の方法で型を設定します。

2つの可能性があります。

1. 変数の型が設定されていない場合、自動的にコンパイラが代わりにタイプを設定します。曖昧でない場合は可能な限り、使用目的から変数型を設定します。例えば:
   ```
   V1:=True
   ```
   コンパイラは変数V1をブール型と決定します。

   同様に次のように書くと:
   ```
   V2:="This is a sample phrase"
   ```
   コンパイラは変数V2をテキスト型と決定します。

   また、上の例ほど明確でない場合でも、コンパイラは以下のように変数のデータ型を設定することができます:
   ```
   V3:=V1  `V3はV1と同じ型
   V4:=2*V2  `V4はV2と同じ型
   ```

   また、4Dのコマンドやメソッドに対する呼び出しからも、コンパイラは変数のデータ型を決定します。例えばブール型と日付型の引数をメソッドに渡した場合、コンパイラは呼び出されたローカル変数$1および$2にブール型と日付型を割り当てます。

   推測からデータ型が決定される場合、環境設定で指定されていない限りは整数、倍長整数、文字列のような制限のあるデータ型が割り当てられることはありません。デフォルトとして常に最も広範囲な型が割り当てられます。例えば:
   ```
   Number:=4
   ```
   と記述した場合、ここでは4が整数であっても他の状況では4.5になる可能性もあるので、Numberには実数型が割り当てられます。

   変数の型を整数、倍長整数または文字列にする場合は、コンパイラ指示子を使います。これらのデータ型はメモリ占有率が少ないため、処理速度が早くなります。

   すでに変数型を設定済みで、定義の整合性も完全であると確信した場合、コンパイラの環境設定を使用して、コンパイラに型の自動決定を行わないようコ ンパイラに指示することができます。定義が徹底されておらず、不完全な場合、コンパイル終了時にエラーメッセージが表示され、必要な変更を行うよう求めら れます。
2. コンパイラ指示子を使用すると、データベースで使用される変数の型を明示的に定義できます。使用方法は次の通りです:
   ```
   C_BOOLEAN(Var)
   ```

   このような指示子を通じて、コンパイラにブール型の変数を作成するよう通知できます。

   アプリケーションでコンパイラコマンドが使用されていれば、コンパイラが型を推測するという作業をしなくて済みます。

   コンパイラ指示子は、代入や用途から得られた結果より優先されます。

   コンパイラ指示子C_INTEGERで定義した変数は、実際にはC_LONGINTで定義したものと同じです。これらは実際には、-2147483648から+2147483647までの倍長整数です。

開発者が型定義する変数

コンパイラに型定義をチェックさせたくない場合、コンパイラ指示子を識別できるようなコードをコンパイラに与える必要があります。
これを行う方法は以下です:
プロセス変数、インタープロセス変数についてのコンパイラ指示子および引数を、名前の先頭がキーワード**Compiler**で始まる1個あるいは複数個のメソッドに記述します。デフォルトで、コンパイラに5種類のコンパイラメソッドを自動生成させ、変数、配列、メソッド引数ごとに命令をまとめることができます（詳細はDesign Referenceマニュアルを参照）。

Note: 引数定義のためのシンタックスは次のとおりです:
Directive (methodName;parameter). このシンタックスをインタプリタモードで実行することはできません。

特別な引数

- **データベースメソッドが受け取る引数** これらの引数が明確に定義されていなければ、コンパイラが型を決定します。定義する場合は、そのデータベースメソッド内で行わなければなりません。この引数定義をCompilerメソッド内に記述することはできません。
  例：は6つのテキスト引数$1から$6までを受け取ります。データベースメソッドの最初に、C_TEXT**($1;$2;$3;$4;$5;$6)**を記述する必要があります。
- **トリガ**
  トリガの結果である引数$0（倍長整数）は、引数が明確に定義されていなければ、コンパイラが型を決定します。定義する場合は、トリガ自身の中で行う必要があります。この引数定義は、Compilerメソッド内には記述できません。
- **"On Drag Over"フォームイベントを受け入れるオブジェクト**
  "On Drag OVer"フォームイベントの結果である引数$0（倍長整数）は、引数が明確に定義されていなければ、コンパイラが型を決定します。定義する場合は、オブジェクトメソッドの中で行う必要があります。この引数定義は、Compilerメソッド内には記述できません。

Note: コンパイラは引数$0を初期化しません。したがって、On Drag Overフォームイベントを使用したら、直ちに"$0"を初期化しなければなりません。例えば:

```
C_LONGINT($0)
If(Form event=On Drag Over)
   $0:=0
   ...
   If($DataType=Is Picture)
      $0:=-1
   End if
   ...
End if
```

型の矛盾として見なされない場合

コンパイラ指示子によって、曖昧なデータ型は除外されます。特定の厳密さは要求されますが、コンパイラが全ての不一致を受け入れないというわけではありません。

例えば整数と定義した変数に実数値を代入すると、コンパイラはこれを型の不一致と見なさず、コンパイラ指示子に応じた値を代入します。以下のように記述した場合:

```
C_INTEGER(vInteger)
vInteger:=2.5
```

コンパイラはこれを型の矛盾としては扱わず、したがってコンパイルは可能です。そして代わりに数値の小数部分を切り上げます（2.5ではなくて3になります）。

型指定ガイド

この節では変数型に生じる矛盾の主な原因と問題を解決するための方法について説明します。

単純変数における型の矛盾

単純なデータ型の矛盾には以下のパターンがあります:

- 2種類の用途による矛盾
- 用途とコンパイラコマンドの矛盾
- 暗黙の型変更による矛盾
- 2つのコンパイラコマンドによる矛盾

2種類の用途による矛盾

最も単純な型の矛盾は、1つの変数名が2つの異なるオブジェクトを示す場合です。例えば、以下のように書いたとします。

```
Variable:=5
```

また、同じアプリケーションのどこかで次のように書いたとします。

```
Variable:=True
```

この記述は、データ型の矛盾を引き起こします。これはどちらか一方の変数名を変えることで解決できます。

コンパイラ指示子との矛盾

例えば、以下のように書き、

```
Variable:=5
```

また、同じアプリケーション内で、次のように書いたとします。

```
C_BOOLEAN(Variable)
```

コンパイラ指示子が最初に処理されるので、Variableはブール型に設定されますが、

```
Variable:=5
```

というステートメントがあるため、これを型の矛盾と見なします。変数名かコンパイラコマンドのどちらかを変更することで、問題は解決します。

1つの式の中で異なるデータ型の変数を使うと矛盾が生じます。コンパイラはそれを型の不一致として指摘します。簡単な例を紹介します。

```
vBool:=True `コンパイラはvBoolをブール型と推測します。
C_INTEGER(<>vInteger) `コンパイラ指示子により整数として宣言されます。
<>vInteger:=3 `コンパイラ指示子と互換のある記述
Var:=<>vInteger+vBool `この操作は、式に非互換のデータ型が混在している
```

暗黙の型変更による矛盾

関数の中には戻り値のデータ型が明確に決まっているものがあります。このような関数の戻り値を不注意に異なった型の変数に代入すると、データ型の矛盾が起ります。

例えば、インタプリタで以下のように書いた場合、

```
IdentNo:=Request("Identification Number") `IdentNo はテキスト型
If(Ok=1)
   IdentNo:=Num(IdentNo) `IdentNo は実数型
   QUERY([Contacts]Id=IdentNo)
End if
```

この例では、3行目で型の矛盾が生じます。これを解決するためには変数の動作を制御する方法が考えられます。異なる名前の中間変数を作成する必要がある場合や、次の例のようにメソッドの構造を変更することで修正できる場合もあります。

```
IdentNo:=Num(Request("Identification Number")) `IdentNo は実数型
If(Ok=1)
   QUERY([Contacts]Id=IdentNo)
End if
```

2つのコンパイラ指示子による矛盾

同じ変数に対して2つの異なるコンパイラ指示子を使用すると、型の再定義になります。例えば、1つのデータベースで次のように書いたとします。

```
C_BOOLEAN(Variable)
C_TEXT(Variable)
```

この場合、コンパイラによって矛盾が検出され、エラーファイルに出力されます。通常、どちらか一方の変数名を変えることで問題は解決します。

C_STRINGを使用する場合、最大の文字長を変更することでデータ型の矛盾が生じることがあるので、注意してください。例えば、次のように書くと、コンパイラは型の矛盾と見なします。

```
C_STRING(5;MyString)
MyString:="Hello"
C_STRING(7;MyString)
MyString:="Flowers"
```

文字列型の変数を定義する際には、十分なサイズのエリアが必要ですが、デフォルトによりコンパイラは短い方を採用してしまいます。解決するには、文字列に最大値を与えるためのコンパイラコマンドを使用することができます。以下のように書くことができます。

```
C_STRING(7;String)
String:="Flowers"
String:="Hello"
```

注：以下のようにC_STRING(7; String)を2回記述することは可能です。

```
C_STRING(7;String)
String:="Flowers"
C_STRING(7;String)
String:="Hello"
```

コンパイラは上記の記述を受け入れますが、2番目の指示子は余分です。

ローカル変数についての注意

ローカル変数におけるデータ型の矛盾は、プロセス変数やインタープロセス変数とほとんど同じものです。唯一の違いは、指定されたメソッドの中のみ型が一貫していればよいという点です。
プロセス変数やインタープロセス変数では、データベースの全体のレベルで型の矛盾が起きましたが、ローカル変数の場合は、メソッドレベルで矛盾が起きます。
例えば、同じメソッドの中で、

```
$Temp:="Flowers"
```

を記述し、さらに

```
$Temp:=5
```

を記述することはできません。

しかし、

```
$Temp:="Flowers"
```

をメソッド1で記述し、

```
$Temp:=5
```

をメソッド2で記述することは出来ます。これは、ロカール変数の有効範囲が同じメソッド内のみであり、データベース全体ではないからです。

配列の矛盾

型の矛盾に配列の要素数は無関係です。コンパイル前のデータベースと同様、配列も動的に管理されます。配列の要素数はメソッドを通して変更可能で、配列用に最大値を定義する必要はありません。このため、要素数を0にすることも、追加や消去、もしくは内容の削除を行うことも可能です。

コンパイルを前提にデータベースを作る場合は、以下の原則に従ってください。

- 配列要素のデータ型を変えないこと
- 配列の次元数を変えないこと
- 文字列では、文字の長さを変えないこと

配列要素のデータ型の変更

整数として定義された配列は、データベース全体を通して整数配列でなければなりません。データベース内で、ブール型などの要素を使用することはできません。
例えば、

```
ARRAY INTEGER(MyArray;5)
ARRAY BOOLEAN(MyArray;5)
```

と書いた場合、MyArrayの型を決定できません。どちらか一方の配列名を変える必要があります。

配列の次元数の変更

コンパイル前のデータベースでは、配列の次元数を変更することができます。コンパイラでシンボルテーブルを作成する際、1次元配列と2次元配列は別々に管理されます。このため、1次元配列を2次元配列として、もしくは2次元配列を1次元配列として再定義することはできません。したがって、同じデータベースの中で以下のように定義することはできません。

```
ARRAY INTEGER(MyArray1;10)
ARRAY INTEGER(MyArray1;10;10)
```

ただし、同じアプリケーションの中に次のステートメントを書くことはできます。

```
ARRAY INTEGER(MyArray1;10)
ARRAY INTEGER(MyArray2;10;10)
```

データベース内で配列の次元数を変更することはできませんが、配列のサイズを変更できます。2次元配列の1つの配列のサイズを変更するには次のように書きます。

```
ARRAY BOOLEAN(MyArray;5)
ARRAY BOOLEAN(MyArray;10)
```

注：2次元配列は実際には複数の1次元配列から構成されています。詳細は、の節を参照してください。

固定長文字配列の場合

文字列には固定長文字列と同じ規則が適用されます。

次のように書いた場合、

```
ARRAY STRING(5;MyArray;10)
ARRAY STRING(10;MyArray;10)
```

コンパイラにより長さの不一致が検出されます。これは最大の文字列の長さを定義することで簡単に解決できます。短い方の文字列は、コンパイラが自動的に処理します。

暗黙の型変更

コマンドのCOPY ARRAY、LIST TO ARRAY、ARRAY TO LIST、SELECTION TO ARRAY、SELECTION RANGE TO ARRAY、ARRAY TO SELECTION、またはDISTINCT VALUESの使用により、データ型の要素、次元数、もしくは文字列や文字の長さが（意図的にもしくは無意識的に）変更されてしまう場合があります。その場合、前に述べた3つの状況のいずれかに該当します。
コンパイラはエラーメッセージを出力します。通常、修正が必要な点は非常に明確です。
暗黙の配列の型変更の例は、の節を参照してください。

ローカル配列

ローカル配列（これを生成したメソッドによってのみ見ることができる配列）が含まれるデータベースをコンパイルする場合、使用前に4Dで明示的にこの配列を定義する必要があります。明示的に配列を定義するとは、ARRAY REALコマンドやARRAY INTEGERコマンドなどを使い、配列を定義することをいいます。メソッドにより10要素のローカル整数配列を生成する場合、メソッドの中に次の行が必要です。

```
ARRAY INTEGER($MyArray;10)
```

フォーム内に作成された変数の型

フォーム内で作成された変数型（例えば、ボタン、ドロップダウンリストボックスなど）は、すべてプロセスもしくはインタープロセス変数です。

インタプリタのデータベースでは、フォーム変数のデータ型は重要ではありませんが、コンパイルされたアプリケーションの場合は重要です。ただし、規則は非常に明確です。

- コンパイラコマンドでフォーム変数の型を定義します、もしくは
- 環境設定（Design Referenceマニュアルを参照）のコンパイルオプションでデフォルト型を割り当てるように設定することができます。

デフォルトで実数とされる変数

次のフォーム変数は、デフォルトとして実数型として設定されます。

チェックボックス
3Dチェックボックス
ボタン
ハイライトボタン
透明ボタン
3Dボタン
ピクチャボタン
グリッドボタン
ラジオボタン
3Dラジオボタン
ラジオピクチャ
ピクチャメニュー
階層ポップアップメニュー
階層リスト
ルーラ
ダイヤル
サーモメータ

注：ルーラ、ダイアルおよびサーモメータのフォーム変数については、例えば環境設定のボタン型に倍長整数を選んだとしても、常に実数型に設定されます。

上記いずれかの変数に対して、データ型の矛盾が起こるとすれば、それは同一の変数名がデータベース内に存在する場合です。この場合は、いずれか1つの変数名を変更してください。

グラフ変数

グラフエリアは自動的にグラフ（倍長整数）のデータ型に設定されるので、型の不一致が生じることはありません。もしデータ型の矛盾が起こるとすれば、それは同一の変数名がデータベース内に存在する場合です。この場合は、いずれか1つの変数名を変更してください。

プラグインエリア変数

プラグインエリアは常に倍長整数型に設定されるので、型の不一致が生じることはありません。もしデータ型の矛盾が起こるとすれば、それは同一の変数名がデータベース内に存在する場合です。この場合は、いずれか1つの変数名を変更してください。

デフォルトでテキストとされる変数

テキスト型に設定されるフォーム変数は以下のとおりです。

入力不可変数
入力可変数
ドロップダウンリスト
メニュー、ドロップダウンリスト
スクロールエリア
コンボボックス
ポップアップメニュー
タブコントロール

これらの変数は2つのカテゴリーに分類できます。

- 単純変数：入力可、入力不可の変数
- 表示変数：ドロップダウンリスト、メニュー/ドロップダウンリスト、スクロールエリア、ポップアップメニュー、コンボボックスおよびタブコントロール

- 単純変数
  デフォルトのデータ型はテキストです。メソッドやオブジェクトメソッドで使用される場合は、ユーザが選択したデータ型が割り当てられます。同じ名前で異なる型の変数が存在する場合以外は、データ型の矛盾が起こることはありません。

- 表示変数
  変数はフォーム内で配列を表示するために使用されます。フォームエディタにデフォルト値が設定されている場合、配列を定義するためのARRAY STRINGコマンドやARRAY TEXTコマンドを使い、対応する変数を明確に定義する必要があります。

ポインタ

ポインタを使うと、4Dツールの強力で多彩な機能を活用することができます。
コンパイル後もポインタの利点をそのまま活用できます。1つの ポインタでデータ型の異なる変数を指定することができます。1つの変数に異なるデータ型を代入して矛盾を引き起こさないようにしてください。また、ポインタで参照する変数のデータ型を変更しないよう注意してください。
この問題の例を示します。

```
Variable:=5.3
Pointer:=->Variable
Pointer->:=6.4
Pointer->:=False
```

この例では、ポインタで参照する変数の型は実数です。これにブール値を代入しているので、データ型の矛盾が起こります。
1つのメソッド内で、異なった目的のためにポインタを使う場合には、参照先の変数型を決定してください。

```
Variable:=5.3
Pointer:=->Variable
Pointer->:=6.4
Bool:=True
Pointer:=->Bool
Pointer->:=False
```

ポインタは、常に参照するオブジェクトとの関連で定義されています。このため、ポインタによるデータ型の矛盾をコンパイラが検知することはできません。矛盾 があっても、変数設定フェーズやコンパイルフェーズでは、エラーメッセージが出力されません。ただし、ポインタに関係する矛盾を見つける方法がまったくない わけではありません。コンパイラの環境設定の**範囲チェック**オプションでポインタの使用状況を確認することができます（詳細はDesign Referenceマニュアルを参照してください）。

プラグインコマンド

一般的なポイント

コンパイルの際、コンパイラはデータベースで使用されるプラグインコマンドのパラメータの数や型定義などについて解析します。メソッドからのプラグイン呼び出しが、プログラム記述の段階で一致していれば、矛盾が生じる危険はありません。

プラグインは、**PlugIns** フォルダに入れることでインストールします。フォルダは、4Dに許可されている次のいずれかの場所に置きます。
ストラクチャファイルと同階層または、Windowsでは実行可能なアプリケーションの隣、またはMacintoshではソフトウェアパッケージ内です。互換性のためにストラクチャと同階層のWin4DXまたはMac4DXフォルダを使用することも出来ます。詳細はインストールガイドを参照してください。

コンパイラは、これらのファイルを複製しませんが、これらを分析して、これらのルーチンの適切な定義を決定します。
プラグインを見つけることができない場合、コンパイラはファイルを開くダイアログボックスを表示してプラグインの場所を確認します。

暗黙の引数を受け取るプラグインコマンド

特定のプラグイン（例えば4D Write）コマンドの中には、暗黙に4Dコマンドを呼び出すものがあります。例えば4D Writeの場合、WR ON EVENTコマンドのシンタックスは次のようになります。
WR ON EVENT(area;event;eventMethod)

最後の引数は、4Dで作成したメソッドの名前です。このメソッドはイベントが受け取られるたびに4D Writeによって呼び出され、自動的に次の引数を受け取ります。

引数	型	説明
$0	倍長整数	戻り値
$1	倍長整数	4D Write エリア
$2	倍長整数	Shift key
$3	倍長整数	Alt key (Windows); Option key (Mac OS)
$4	倍長整数	Ctrl key (Windows), コマンド key (Mac OS)
$5	倍長整数	イベント型
$6	倍長整数	イベント引数の値

コンパイラがこれらの引数の存在を認識し、これらを考慮するためには、引数をコンパイラコマンド、またはメソッド内での使用法によって型設定しなければなりません。メソッドで使用する場合には、型を明確に推測できるように使用する必要があります。

4Dコンポーネント

4Dを使い、4Dコンポーネントの作成および管理が可能です。(マトリクスデータベースと呼ばれる) 4Dコンポーネントは一連の4Dオブジェクトで、ストラクチャファイルにいくつかの機能がまとめられています。コンポーネントは他のデータベース (ホストデータベース) にインストールできます。

インタプリタモードで動作するホストデータベースは、インタプリタおよびコンパイル済みのコンポーネントを使用できます。一つのホストデータベース にインタプリタおよびコンパイル済みのコンポーネントを同時にインストールできます。他方、コンパイルモードのホストデータベースは、インタプリタモード のコンポーネントを使用できません。コンパイル済みのコンポーネントのみが使用できます。

インタプリタモードのコンポーネントを含むインタプリタのホストデータベースは、インタプリタモードのコンポーネントメソッドを呼ばない限り、コンパイルできます。もし呼び出している場合は、警告ダイアログが表示され、コンパイルに失敗します。

コンポーネントの共有メソッドがホストデータベースのプロジェクトメソッドと同じ名前の場合、名前の衝突が発生します。この場合、ホストデータベー スのコンテキストでコードが実行される際、ホストデータベースのメソッドが呼び出されます。これはコンポーネントのメソッドをカスタムメソッドでマスクす ることが可能であることを意味します (例えば異なる機能を実装するために)。コンポーネントのコンテキストでコードが実行されるときは、コンポーネントのメソッドが呼び出されます。ホスト データベースをコンパイルする際、このマスクは警告として記録されます。

二つのコンポーネントが同じ名前の共有メソッドを公開している場合は、ホストデータベースのコンパイルの際、エラーが生成されます。

コンポーネントに関する詳細は、Design Referenceマニュアルを参照してください。

ローカル変数$0...$Nの処理および引数の受け渡し

ローカル変数の処理はこれまでに述べた規則に従います。他の変数と同様、メソッドの実行中にデータ型を変更することはできません。ここでは、型の矛盾を起こす可能性のある2つのケースについて検証します。

- 型変更が必要な場合。ポインタの使用によりデータ型の矛盾を避けることができます。
- 引数の間接参照が必要な場合

ポインタの使用で型の矛盾を避ける

変数型を変更することはできませんが、ポインタを使用して異なるデータ型を参照することはできます。例えば、1次元配列 のメモリサイズを返す関数を考えてみてください。この場合、メモリサイズを数字で表せないテキスト配列とピクチャ配列を除き、実数の結果のみを返します（の節を参照してください）。
また、テキスト配列とピクチャ配列については、文字列の結果を返します。この関数の引数は、メモリサイズを調べようとする配列へのポインタです。この操作を行うには2つの方法があります。

- ローカル変数のデータ型を気にせずに使用する方法。このメソッドはインタプリタでしか動作しません。
- ポインタを使用し、インタプリタおよびコンパイルモードで処理する方法。

MemSize機能のインタプリタモードでのみ動作する例（Macintoshを使用する場合の例）

```
$Size:=Size of array($1->)
$Type:=Type($1->)
Case of
   :($Type=Real array)
      $0:=8+($Size*10) ` $0は実数
   :($Type=Integer array)
      $0:=8+($Size*2)
   :($Type=LongInt array)
      $0:=8+($Size*4)
   :($Type=Date array)
      $0:=8+($Size*6)
   :($Type=Text array)
      $0:=String(8+($Size*4))+("+Sum of text lengths") ` $0はテキスト
   :($Type=Picture array)
      $0:=String(8+($Size*4))+("+Sum of picture sizes") ` $0はテキスト
   :($Type=Pointer array)
      $0:=8+($Size*16)
   :($Type=Boolean array)
```

```
      $0:=8+($Size/8)
 End case
```

このメソッドでは、$0のデータ型が$1の配列によって異なるため、コンパイルできません。

MemSize機能のインタプリタモードおよびコンパイルモードで動作可能なバージョン（Macintoshを使用する場合の例）
これはポインタを使用する方法です。

```
 $Size:=Size of array($1->)
 $Type:=Type($1->)
 VarNum:=0
 Case of
    :($Type=Real array)
       VarNum:=8+($Size*10) ` VarNumは実数1
    :($Type=Integer array)
       VarNum:=8+($Size*2)
    :($Type=LongInt array)
       VarNum:=8+($Size*4)
    :($Type=Date array)
       VarNum:=8+($Size*6)
    :($Type=Text array)
       VarText:=String(8+($Size*4))+("+Sum of text lengths")
    :($Type=Picture array)
       VarText:=String(8+($Size*4))+("+Sum of picture sizes")
    :($Type=Pointer array)
       VarNum:=8+($Size*16)
    :($Type=Boolean array)
       VarNum:=8+($Size/8)
 End case
 If(VarNum#0)
    $0:=->VarNum
 Else
    $0:=->VarText
 End if
```

2つの関数は、以下の点が違っています。

- 1番目の関数では、結果が変数であること
- 2番目の関数では、結果が変数へのポインタであること。結果はポインタで簡単に参照できます。

引数の間接参照

コンパイラは、引数の間接参照の機能をサポートしています。インタプリタでは、引数のデータ型と数値を自由に設定できます。データ型の矛盾を起こさない限り(呼ばれた側でセットされていない引数を使用しないこと)、コンパイルモードでも同様に自由に設定することができます。

型の矛盾の元になるので、間接参照する引数は、すべて同じデータ型でなければなりません。

この間接参照は以下の条件を守ることにより、正しく動作します：引数の一部のみを間接参照する場合、直接参照の引数の後に間接参照引数を配置するようにします。

メソッド内で、間接参照は${$i}のように表示します。$iは数値変数です。${$i}を「ジェネリックパラメータ（generic parameter）」と呼びます。

以下は間接参照の例です。数値を合計し、引数として渡されたフォーマットに編集して返すような関数を考えてください。合計される元の数値は、メソッドが呼ばれるたびに変わります。このメソッドでは数値と編集フォーマットを引数としてメソッドに渡さなければなりません。
この関数は、以下のようにして呼び出します。

```
 Result:=MySum("##0.00";125,2;33,5;24)
```

この場合、数値を合計し、指定したフォーマットに編集された"182.70"が返されます。関数の引数は正しい順序で渡してください。最初にフォーマット、次に値です。

以下はMySum関数です。

```
 $Sum:=0
 For($i;2;Count parameters)
    $Sum:=$Sum+${$i}
 End for
 $0:=String($Sum;$1)
```

この関数は次の方法で呼び出すことができます。

```
 Result:=MySum("##0.00";125,2;33,5;24)
 Result:=MySum("000";1;18;4;23;17)
```

他のローカル変数と同様、ジェネリックパラメータはコンパイラに指示する必要はありません。ただし、曖昧になりそうな場合や最適化のために必要な場合は以下のシンタックスを使用することができます。

```
 C_INTEGER(${4})
```

このコマンドは、4番目以降に間接参照されたすべての引数のデータ型が整数であることを意味します。$1、$2、$3には、いかなるデータ型も使用できますが、$2を間接参照した場合には影響を受けます。このため、実数であってもデータ型は整数と見なされます。

注：コンパイラはこのコマンドを変数設定フェーズで使います。定義に使用する数値は変数ではなく、定数でなくてはなりません。

予約変数と定数

4Dの変数および定数には、コンパイラによってデータ型および識別子が割り当てられているものがあります。このため、ユーザはこれらの変数名や定数を、新しい変数やメソッド、関数、プラグインコマンドに使用することはできません。インタリタモードで行うのと同じ方法で、これらの値を検証し使用することができます。

システム変数

4Dのおよび対応するデータ型の完全なリストを紹介します。

変数	型
OK	倍長整数
Document	テキスト
FldDelimit	倍長整数
RecDelimit	倍長整数
Error	倍長整数
Error method	テキスト
Error line	倍長整数
MouseDown	倍長整数
KeyCode	倍長整数
Modifiers	倍長整数
MouseX	倍長整数
MouseY	倍長整数
MouseProc	倍長整数

クイックレポート変数

クイックレポートで計算用のカラムを作成する際、最初のカラムに変数C1を2番目にはC2を、そして3番目のカラムにはC3をというように4Dが自動的に変数を作成します。この処理はユーザには見えません。
これらの変数をメソッド内で使用する場合、他の変数と同様C1、C2...Cnは型変更することができませんので、留意してください。

4Dの定義済み定数

4Dで定義されている定数は定数テーマリストを使用して見つけることができます。4Dの定数は、デザインモードのエクスプローラーにも表示されます。

シンタックスの詳細

コンパイラは、4Dコマンドの通常の構文ルールに従っていることを期待します。コンパイルのためにデータベースを特に変更する必要はありません。

とはいえ、この節ではコマンドについての注意点や詳細について項目別に説明します。

- 変数のデータ型を決定付けるようなコマンドは、データ型矛盾の原因になることがあります。
- コマンドの中には複数のシンタックスを持つものがあり、どのシンタックスが最適なのか知っておくと役に立ちます。

文字列

Character code (character)

文字列に対して処理を行うコマンドでは、Character code関数のみが特段の注意を必要とします。インタプリタモードでは、空でない文字列や空の文字列をこの関数に渡せます。
コンパイルモードでは、空の文字列を渡すことができません。
Character codeに、空の文字列が格納された変数を渡すと、コンパイラはコンパイル中にエラーを見つけることはできません。

通信

SEND VARIABLE(variable)
RECEIVE VARIABLE(variable)

上記2つのコマンドは変数をディスクに保存または読み込む場合に使用されます。変数が引数としてこれらのコマンドに渡されます。
コマンドに渡す変数の型は、常に同じ型でなければなりません。変数のリストをファイルに送る場合を考えてみましょう。誤ってデータ型を変えてしまう恐れを取り除くため、送信する変数のデータ型をリストの先頭で指定することをおすすめします。そうすると、これらの変数を受け取る際には、常に最初にインジケータが返されることになります。RECEIVE VARIABLEコマンドを呼び出したら、Case of文を使用して、次から受け取るデータを処理できます。

例

```
SET CHANNEL(12;"File")
If(OK=1)
   $Type:=Type([Client]Total_TO)
   SEND VARIABLE($Type)
   For($i;1;Records in selection)
      $Send_TO:=[Client]Total_TO
      SEND VARIABLE($Send_TO)
      NEXT RECORD
   End for
End if
SET CHANNEL(11)
SET CHANNEL(13;"MyFile")
If(OK=1)
   RECEIVE VARIABLE($Type)
   Case of
      :($Type=Is String Var)
         RECEIVE VARIABLE($String)
  `受信した変数の処理
      :($Type=Is Real)
         RECEIVE VARIABLE($Real)
  `受信した変数の処理
      :($Type=Is Text)
         RECEIVE VARIABLE($Text)
  `受信した変数の処理
   End case
End if
SET CHANNEL(11)
```

ストラクチャアクセス

Field(フィールドポインタ) または (テーブル番号;フィールド番号)
Table(テーブルポインタ) または (テーブル番号) または (フィールドポインタ)

これら2つのコマンドは、与えられた引数によって、戻り値のデータ型が異なります。

- ポインタを与えると、Table関数は数値を返します。
- 数値を与えると、Table関数はポインタを返します。

コンパイラでは、これらの関数から結果のデータ型を決定できません。このような場合は、コンパイラコマンドを使用して明確に定義してください。

ドキュメント

Open document、Append document、Create Document関数によって返されるドキュメント参照番号のデータ型は時間型です。

演算

Mod **(value;divider)**

4Dでは25を3で割った余りを求める場合、次の2通りの方法があります。

```
Variable:=Mod(25;3)
```

または

```
Variable:=25%3
```

コンパイラはこの2つの式を区別します。Mod関数はすべての数値に使用できますが、%演算子は整数と倍長整数にしか使用できません。%演算子のオペランドが、倍長整数データ型の範囲を越えた場合には、返される結果は保証されません。

例外処理

IDLE
ON EVENT CALL (Method{; ProcessName})
ABORT

ON EVENT CALL

IDLEコマンドは例外処理を行うために4Dランゲージに追加されました。ON EVENT CALLコマンドを用いる場合は、必ずこのIDLEコマンドも使用してください。

このコマンドはイベント管理命令として定義することができます。

4Dのカーネルだけがシステムイベント（マウスクリックやキー操作など）を検知できます。ほとんどの場合、カーネルコールはコンパイル後のコードそのものによって起動され、ユーザに対しては透過的です。

他方、4Dがイベント待ちループなどの中でイベントの受信を待っている場合はカーネルコールがないことが明白です。

Windows版の例

```
  `「MouseClick」メソッド
If(MouseDown=1)
   <>vTest:=True
   ALERT("マウスクリックされました。")
End if

  `「Wait」メソッド
<>vTest:=False
ON EVENT CALL("MouseClick")
While(<>vTest=False)
  `イベント待ちのループ
End while
ON EVENT CALL("")
```

この場合、IDLEコマンドを次のように追加します。

```
  `「Wait」メソッド
<>vTest:=False
ON EVENT CALL("MouseClick")
While(<>vTest=False)
   IDLE
  `イベントを検知させるためのカーネルコール
End while
ON EVENT CALL("")
```

ABORT

このコマンドは、エラー処理プロジェクトメソッド内でのみ使用してください。これは4Dで使用した場合とまったく同じように動作しますが、EXECUTE FORMULA、APPLY TO SELECTION、APPLY TO SUBSELECTIONコマンドから呼び出されたメソッド内の場合は例外です。このような状況は避けた方がよいでしょう。

配列

コンパイラが配列のデータ型を決める際に使用する4Dコマンドは、以下の7種類です。

COPY ARRAY(source;destination)
SELECTION TO ARRAY(フィールド;配列)
ARRAY TO SELECTION(配列;フィールド)
SELECTION RANGE TO 配列(start;end;フィールド;配列)
LIST TO ARRAY(list;配列{; itemRefs})
ARRAY TO LIST(配列;list{; itemRefs})
DISTINCT VALUES(フィールド;配列)

COPY ARRAY

COPY ARRAYコマンドは2個の配列型の引数を使用します。引数の一方がどこにも定義されていないと、コンパイラは定義されている方のデータ型から未定義の配列のデータ型を決定します。
この場合、以下のように処理されます。

- 最初の引数が定義されている場合：2番目の配列には、最初の配列のデータ型が適用されます。
- 2番目の引数が定義されている場合：最初の配列に2番目の配列のデータ型が適用されます。

コンパイラはデータ型を厳密にチェックするので、COPY ARRAYコマンドは同じ型の配列間で行わなければなりません。このため、整数と倍長整数と実数、あるいは、テキスト配列と文字配列で文字列の長さが一定でない場合等のように、型の似ている配列間のコピーをする際には要素を1つずつコピーする必要があります。
例えば、整数型の配列要素を実数型の配列にコピーする場合、次のように行います。

```
$Size:=Size of array(ArrInt)
ARRAY REAL(ArrReal;$Size)
  `実数配列を整数配列と同じ配列サイズ（要素数）にする。
For($i;1;$Size)
   ArrReal{$i}:=ArrInt{$i}
  `個々の要素についてコピーを行う。
End for
```

処理中に配列の次元数を変更することはできませんので、注意してください。1次元配列を2次元配列にコピーすると、エラーメッセージが出力されます。

SELECTION TO ARRAY, ARRAYTO SELECTION, DISTINCT VALUES, SELECTION RANGE TO ARRAY

4Dのインタプリタモードと同様、これら4つのコマンドでは配列の定義は要求されません。型が定義されていない配列にはコマンドで指定した フィールドのデータ型が割り当てられます。
例

```
SELECTION TO ARRAY([MyTable]IntField;MyArray)
```

"IntField"が整数フィールドの場合、"MyArray"は整数配列になります。

配列が定義されている場合は、フィールドと同じデータ型になっているかどうか確認してください。整数、倍長整数、実数は似ていますが、同じ型ではありません。
ただし、テキストや文字のデータ型では、多少許容範囲が広くなります。デフォルトにより、文字型のフィールドを使用するコマンドで配列をあらかじめ定義せずに引数として使った場合、配列に割り当てられるデフォルトのデータ型はテキストです。あらかじめ配列を文字またはテキスト型として宣言されている場合、ユーザの指定が適用されます。

テキスト型のフィールドについても同様で、宣言された型が優先されます。
SELECTION TO ARRAY、SELECTION RANGE TO ARRAY、ARRAY TO SELECTION、DISTINCT VALUESコマンドは、1次元の配列でしか使用できません。
SELECTION TO ARRAYコマンドにはもう1つのシンタックスがあります。
SELECTION TO ARRAY(テーブル;配列).
この場合、配列変数は倍長整数になります。SELECTION RANGE TO ARRAYコマンドについても同様です。

LIST TO ARRAY, ARRAY TO LIST

LIST TO ARRAYおよびARRAY TO LISTコマンドに使用できる配列は次の2種類だけです。

- 1次元の文字配列
- 1次元のテキスト配列

このコマンドの場合、引数に使う配列をあらかじめ宣言する必要はありません。デフォルトでは、宣言されていない配列は、テキスト配列になります。配列があらかじめテキストか文字型で宣言されていない場合、ユーザの指定した型になります。

配列関連コマンドでのポインタ使用

ポインタについての節で説明したように、配列を定義するコマンドの引数にポインタ参照が使われていると、コンパイラは型の矛盾を発見できません。

```
SELECTION TO ARRAY([Table]Field;Pointer->)
```

Pointer->が示すのが配列だとして、コンパイラは配列の型およびフィールド型をチェックすることができません。型の矛盾が起こらないようにするのは、開発者は気をつけるべきです。ポインタが示す配列を必ず定義してください。
コンパイラは、引数にポインタを使用している配列定義ステートメントを見つけると、警告メッセージを出力します。警告メッセージはこの種の矛盾を見つける際に役立ちます。

ローカル配列

データベースで、ローカル配列（定義されたメソッド内のみで有効な配列）を使用している場合は、使用前に明確に宣言しておく必要があります。

ローカル配列を定義するには、ARRAY REAL、ARRAY INTEGERなど、配列を定義するコマンドを使用します。

例えば、プロシージャで10個の要素を持つローカルな整数配列を作る場合、次のようなコマンドを使用前に定義しておきます。

```
ARRAY INTEGER($MyArray;10)
```

ランゲージ

Get pointer(varName)
Type (object)
EXECUTE FORMULA(statement)
TRACE
NO TRACE

Get pointer

ポインタの配列を初期化する場合、配列の各要素はそれぞれ与えられた変数を表します。
例えばV1、V2、...V12というような12個の変数の場合、以下のように書くことができます。

```
ARRAY POINTER(Arr;12)
Arr{1}:=->V1
Arr{2}:=->V2

Arr{12}:=->V12
```

また、以下のように書くこともできます。

```
ARRAY POINTER(Arr;12)
For($i;1;12)
   Arr{$i}:=Get pointer("V"+String($i))
End for
```

この処理が終了すると、各要素が変数V1からV12を指すポインタの配列ができます。

この2つの書き方は、両方ともコンパイルできますが、他の場所で変数V1...V12の型が明らかにされていないと、コンパイラはデータ型を決定できません。そのため、このような変数は別の場所で明示的に使用するか、または定義する必要があります。
明示的に変数を定義する方法は、2通りあります。

- コンパイラコマンドを使用してV1...V12を定義するには以下のように記述します。

```
C_LONGINT(V1;V2;V3;V4;V5;V6;V7;V8;V9;V10;V11;V12)
```

- メソッドでV1...V12に値を代入するには以下のように記述します。

```
V1:=0
V2:=0

V12:=0
```

Type(object)

コンパイルされたデータベースの変数のデータ型は1つだけなので、この関数は不必要に思えるかも知れません。しかし、ポインタを 使用する際、この関数が役立ちます。例えば、ポインタが参照する変数のデータ型を知りたい場合、ポインタの性質上、指されているオブジェクトがわかりにくい場合があります。

EXECUTE FORMULA

EXECUTE FORMULAコマンドは、インタプリタモードでは有効ですが、コンパイルモードではその利点を活かすことができません。
コンパイルモードでは、このコマンドに引数として渡されたメソッド名は解釈されます。コンパイラのいくつかの利点を活用できない上に、引数のシンタックスチェックもできません。
また、引数にローカル変数を使用することもできません。
EXECUTE FORMULAコマンドは、複数のステートメントに置き換えることができます。以下に2つの例を挙げます。

```
i:=FormFunc
EXECUTE FORMULA("FORM SET INPUT (Form"+String(i)+")")
```

これは、以下ように書き換えることができます。

```
i:=FormFunc
VarForm:="Form"+String(i)
FORM SET INPUT(VarForm)
```

また次の例では、

```
$Num:=SelPrinter
EXECUTE FORMULA("Print"+$Num)
```

ここでは、EXECUTE FORMULAコマンドをCase ofに置き換えることができます。

```
Case of
   :($Num=1)
      Print1
   :($Num=2)
      Print2
   :($Num=3)
      Print3
End case
```

EXECUTE FORMULAコマンドは常に置き換えが可能です。実行するメソッドはデータベースのプロジェクトメソッドのリストから選択されたもので、その数には限りがあります。このため、EXECUTE FORMULAコマンドは必ずCase of文や他のコマンドで置き換えることができます。さらに、コードの実行速度はEXECUTE FORMULAコマンドよりも速くなります。

TRACE, NO TRACE

これらのコマンドはデバッグ処理の段階で使用します。コンパイルされたデータベースでは機能しません。コンパイラはこれらのコマンドを無視するので、メソッド内に残しておいても問題ありません。

変数

Undefined(variable)
SAVE VARIABLES(document;variable1{; variable2…})
LOAD VARIABLES(document;variable1{; variable2…})
CLEAR VARIABLE(variable)

Undefined

コンパイラではコンパイルモードの変数は必ず定義されます。データ型は、コンパイルが完了するまでに設定されます。このため、引数が渡されると毎回Undefined関数はFalseを返します。

注：アプリケーションがコンパイルモードで実行されているかどうかはIs compiled mode関数を呼び出して確認してください。

SAVE VARIABLES, LOAD VARIABLES

インタプリタでは、LOAD VARIABLEコマンドの実行後にUndefined関数を使用して変数が未定義かどうか調べることにより、ドキュメントファイルの存在の有無をチェックできます。コンパイル後はこの方法を使用できません。Undefined関数が常にFalseを返すからです。

このテストはインタプリタでもコンパイル後でも次のようにして実行できます。

1. どの変数にとっても無効な値で、ロードする変数を初期化します。
2. LOAD VARIABLEコマンドを実行した後、ロードした変数のうちの1つを初期値と比較します。

メソッドは以下のようになります。

```
Var1:="xxxxxx"
  `"xxxxxx" この値は、LOAD VARIABLESコマンドにより返されることはない。
Var2:="xxxxxx"
Var3:="xxxxxx"
Var4:="xxxxxx"
LOAD VARIABLES("Document";Var1;Var2;Var3;Var4)
If(Var1="xxxxxx")
  `ドキュメントが見つからない。

Else
  `ドキュメントが見つかった。

End if
```

CLEAR VARIABLE

インタプリタモードでは、この関数には次の2つのシンタックスがあります。
CLEAR VARIABLE(variable)
CLEAR VARIABLE("a")

コンパイル後のデータベースでは、1番目のシンタックスCLEAR VARIABLE (variable)は変数を再度初期化します（数値は0に、文字列やテキストは空にする等）。理由は、コンパイル後は未定義の変数が存在しないからです。

したがって、コンパイル後はテキスト、ピクチャ、BLOB、配列型の変数を除き、CLEAR VARIABLEコマンドで変数のメモリを解放することはできません。

配列の場合、CLEAR VARIABLEコマンドは、要素数が0の新しい配列の定義を意味します。
例えば、CLEAR VARIABLE (MyArray) において、MyArrayが整数配列の場合、以下のいずれか同じ意味を持ちます。

```
ARRAY INTEGER(MyArray;0)
  `一次元配列の場合
ARRAY INTEGER(MyArray;0;0)
  `二次元配列の場合
```

2番目のシンタックスCLEAR VARIABLE("a")は、コンパイラが名前ではなくアドレスで変数にアクセスするためコンパイラで使用することはできません。

Pointers with certain commands

以下のコマンドには、共通する特徴が1つあります。これらのコマンドでは、最初に省略可能な引数[テーブル]があり、2番目の引数にポインタを使用することが可能な点です。

ADD TO SET
APPLY TO SELECTION
COPY NAMED SELECTION
CREATE EMPTY SET
CREATE SET
CUT NAMED SELECTION
DIALOG
EXPORT DIF
EXPORT SYLK
EXPORT TEXT
GOTO RECORD
GOTO SELECTED RECORD
GRAPH TABLE
IMPORT DIF
IMPORT SYLK
IMPORT TEXT
INPUT FORM
LOAD SET
LOCKED ATTRIBUTES
ORDER BY
ORDER BY FORMULA
OUTPUT FORM
PAGE SETUP
Print form
PRINT LABEL
QR REPORT
QUERY
QUERY BY FORMULA
QUERY SELECTION
QUERY SELECTION BY FORMULA
REDUCE SELECTION
RELATE MANY
REMOVE FROM SET

コンパイルモードでは、省略可能な[テーブル]引数を扱うことが可能です。しかし、これらのコマンドに最初に渡された引数がポインタの場合、コンパイラはポインタが何を参照しているのかわからないため、コンパイラは、最初の引数をテーブルポインタとして扱ってしまいます。

QUERYコマンドのケースで考えてみましょう。シンタックスは以下のとおりです。
QUERY({Table{;formula{;*}})
formulaの最初の要素はフィールドでなければなりません。
次のように書いた場合、

```
QUERY(PtrField->=True)
```

コンパイラは2番目の要素内でフィールドを表す記号を探します。しかし、"="の記号が見つかった段階で、コンパイラは、これの関数式を判別できないため、エラーメッセージを出力します。

曖昧にならないようにするためには、以下のいずれかのように記述することができます。

```
QUERY(PtrTable->;PtrField->=True)
```

または

```
QUERY([Table];PtrField->=True)
```

テーブルを省略しないことで、曖昧さを避けることができます。

定数

独自の4DK#リソース(定数)を作成する場合は、必ず数値の型を倍長整数(L)もしくは実数(R)に、文字列の型を(S)に設定してください。その他の型では警告メッセージが表示されます。

最適化のヒント

"良いプログラムを作成する"ための決定的な方法を説明するのは難しいことですが、良い構造を持つプログラムの利点を再度ここで強調します。4Dは構造化プログラミングが可能であり、この能力はプログラミングの大きな助けになります。

構造化されたデータベースのコンパイルは、そうでないデータベースのコンパイルと比べ、同じ労力で得られる結果が大きく異なります。例えば、n個のオブジェクトに対する共通メソッドは、同じステートメントで書かれたn個のオブジェクトメソッドより、インタプリタモードでもコンパイルモードでも、はるかに良い結果をもたらすことでしょう。

つまり、プログラミングの質がコンパイルされたコードの品質にも影響を与えるのです。

経験を積むことで、4Dコードを段階的に改善できます。コンパイラを頻繁に使用して間違いを訂正するフィードバックを得、最も効果的な解決方法に到達できます。

この節では、単純な繰り返しの作業にかかる時間を短縮するためのアドバイスや秘訣を紹介します。

コードにコメントを使う

プログラミングテクニックによっては、コードが他の人や自分自身にとって理解しづらいものもあります。このため、詳細にわたるコメントをメソッドに入れることをお勧めします。コメントが多すぎるとインタプリタデータベースでは実行が遅くなりますが、コンパイルされたデータベースにはまったく影響しません。

コンパイラ指示子によるコードの最適化

コンパイラコマンドを使用すると、コードの実行速度がかなり速くなります。記述のされ方から変数のタイプを決定する場合、一番広い範囲をカバーできるタイプを設定します。例えば、変数のタイプをVar:=5というステートメントで定義する場合、整数のみ使用されているのにもかかわらず、コンパイラはタイプを実数に設定します。

数値変数

変数がコンパイラー指示子で型宣言されていない場合、(データベース設定でコンパイル時のデフォルト数値型が設定されていないと) デフォルトで数値変数には実数が割り当てられます。実数は倍長整数よりも計算が遅いので、数値変数が常に整数だと分かっている場合は、コンパイラ指示子C_LONGINTで変数を定義すると効果的です。
例えば、ループのカウンタは常に整数として定義しておくとよいでしょう。

4D関数の中には整数を返すものがあります (Character code, Int関数等)。コンパイラは、このような関数の結果を未定義の変数に代入する場合も、変数のタイプを整数ではなく実数にします。変数が別のタイプの値に使用されないことが明らかな場合は、必ずコンパイラ指示子で変数を宣言してください。

ここで簡単な例を示します。指定された範囲内のランダムな数を返す関数です。

```
$0:=Mod(Random;($2-$1+1))+$1
```

このように記述されていると、コンパイラは$0のタイプを倍長整数ではなく実数に設定してしまうので、メソッドにコンパイラ指示子を使用してください。

```
C_LONGINT($0)
$0:=Mod(Random;($2-$1+1))+$1
```

メソッドの戻り値に使用するメモリスペースも少なく、メソッドの実行速度も速くなります。

もう1つの例を紹介します。2つの変数を倍長整数として定義します。

```
C_LONGINT($var1;$var2)
```

そして、他の2つの変数の合計が3つ目のタイプの定義されていない変数に返されます。

```
$var3:=$var1+$var2.
```

コンパイラは、3つ目の変数$var3を実数とします。結果を倍長整数としたい場合は、倍長整数として明示的に定義しなければなりません。

注：コンパイルモードでの計算結果は、計算結果を受け取る変数のタイプではなく、計算に使用される数値のデータタイプであることに注意してください。

下記の例では、変数が倍長整数として計算されています。

```
C_REAL($var3)
C_LONGINT($var1;$var2)
$var1:=2147483647
$var2:=1
$var3:=$var1+$var2
```

コンパイルモードおよびインタプリタモードの両方で、$var3は-2147483648となります。
しかし、下記の例では、最適化のためにコンパイラは、1の値を整数と見なします。

```
C_REAL($var3)
C_LONGINT($var1)
$var1:=2147483647
$var3:=$var1+1
```

コンパイルモードでは、倍長整数として計算されるため、$var3は-2147483648となります。インタプリタモードでは、実数として計算されるため、$var3は2147483648となります。

ボタンは倍長整数として定義できる具体的なケースです。

文字

文字の値をテストしたい場合、文字そのものではなくて文字のCharacter code値で比較をしてください。通常の文字として比較した場合、英大文字、小文字の区別はされず同一のものとして比較されます。

その他のヒント

二次元配列

二次元配列は、2番目の次元が1番目の次元より大きい方が実行効率が上がります。
例えば、次のように定義された配列は、

```
ARRAY INTEGER(Array;5;1000)
```

次のような配列より効率が高くなります。

```
ARRAY INTEGER(Array;1000;5)
```

フィールド

フィールドを使用して複数の演算を行う場合、変数にフィールドの値を代入して計算をしたほうが、直接フィールドで計算するより効率が良くなります。
以下のメソッドをご覧ください

```
Case of
   :([Client]Dest="New York City")
      Transport:="Messenger"
   :([Client]Dest="Puerto Rico")
      Transport:="Air mail"
   :([Client]Dest="Overseas")
      Transport:="Express mail service"
   Else
      Transport:="Regular mail service"
End case
```

このメソッドは以下のように記述すると実行速度が速くなります。

```
$Dest:=[Client]Dest
Case of
   :($Dest="New York City")
      Transport:="Messenger"
   :($Dest="Puerto Rico")
      Transport:="Air mail"
   :($Dest="Overseas")
      Transport:="Express mail service"
   Else
      Transport:="Regular mail service"
End case
```

ポインタ

フィールドの場合と同じように、ポインタ参照よりも変数を使用する方が速くなります。
ポインタ参照される変数で何回も計算する場合、値を変数に格納すると時間を節約できます。
例えば、ポインタMyPtrがフィールドや変数を指しており、その値を使用して一連のテストをする場合は以下のように記述することができます。

```
Case of
   :(MyPtr->=1)
      Sequence 1
   :(MyPtr->=2)
      Sequence 2

End case
```

このテストは、以下のように記述することで実行速度が上がります。

```
Temp:=MyPtr->
Case of
   :(Temp=1)
      Sequence 1
   :(Temp=2)
      Sequence 2

End case
```

ローカル変数

コードを作成する場合は、できるだけローカル変数を使用してください。ローカル変数には、次の利点があります。

- ローカル変数は、データベースのスペースを多く必要としません。ローカル変数は、メソッド中で使用されると作成され、メソッドの実行が終わると破棄されます。
- 生成されるコードは、ローカル変数（特に倍長整数）のために最適化されます。これはループカウンタに有効です。

エラーメッセージ

ここでは、コンパイラが出力するメッセージについて説明します。メッセージは、以下の数種類になります。

- 警告メッセージ…見落としがちな過ちを避ける手助けをします。
- エラーメッセージ…間違いを正します。
- 範囲チェックメッセージ…4D内で生成されるメッセージ。

警告メッセージ

このメッセージは、コンパイルの過程で出力されます。ここでは、各メッセージを警告の対象となるコード例と共に示します。

注: 警告はエラーと異なります。警告はコードに誤りがないか、開発者に注意を促すために生成されるものです。警告対象のコードに誤りがないことが確かな場合、警告は無視することができます。

COPY ARRAYコマンド中にポインタが存在します。

```
COPY ARRAY(Pointer->;Array)
```

SELECTION TO ARRAYコマンド中にポインタが存在します。

```
SELECTION TO ARRAY(Pointer->;MyArray)
SELECTION TO ARRAY([MyTable]MyField;Pointer->)
```

ARRAY TO SELECTIONコマンド中にポインタが存在します。

```
ARRAY TO SELECTION(Pointer->;[MyTable]MyField)
```

LIST TO ARRAYコマンド中にポインタが存在します。

```
LIST TO ARRAY(List;Pointer->)
```

ARRAY TO LISTコマンド中にポインタが存在します。

```
ARRAY TO LIST(Pointer->;List)
```

配列定義コマンド中にポインタが存在します。

```
ARRAY REAL(Pointer->;5)
```

ARRAY REAL(Array; Pointer->)と書いた場合、このメッセージは出力されません。配列の次元数はデータタイプに影響を与えないからです。ポインタで参照する配列は、事前に定義しておく必要があります。

DISTINCT VALUESコマンド中にポインタが存在します。

```
DISTINCT VALUES(Pointer->;Array)
```

Undefined関数は使用しないでください。

```
If(Undefined(Variable))
```

Undefined関数はコンパイルしたデータベースでは常にFALSEを返します。

このメソッドは、パスワードにより保護されています。

フォームXページ目の自動動作ボタンの名前がありません。
衝突を避けるため、すべてのボタンに名前が必要です。

ポインタの参照先を文字として処理します。Pointer->[[2]]:="a"

文字列のインデックスを数値タイプとして処理します。

```
String[[Pointer->]]:="a"
```

配列のインデックスを実数タイプとして処理します。

```
ALERT(MyArray{Pointer->})
```

プラグインコマンドの呼び出しでパラメータが不足しています。

```
WR SET FONT(Area)
```

注: 以下のタグを使用すると、個別に警告メッセージを有効にしたり無効にしたりできます:
// %W-warning_number で警告を無効にできます。
// %W+warning_number で警告を有効にできます。
この方法で警告を有効にしたり無効にしたりすると、効果はその後に解析されるすべてのコードに効果があります。全体的に警告を無効にしたり有効にしたりしたい場合は、"Compiler_xxx"という名前のメソッドに適切なタグを記述することができます。このような名前を持つメソッドがまずコンパイラーにより解析されるためです。例えば"COPY ARRAY中にポインターがあります"警告を無効にするには、"// %W-518.1"タグを記述します。

エラーメッセージ

このメッセージはコンパイルの処理段階で生成されます。データベースのコンパイルにより表示されるエラーを修正するかどうかはユーザが判断します。各メッセージを問題のあるコードの例と共に示します。
メッセージは次のグループに分けて説明します：タイプ、シンタックス、引数、演算子、プラグインコマンド、全般のエラー

タイプ

変数タイプの演算子に互換性がありません。そのため、タイプを設定することが出来ません。

```
MyReal:=12.3
MyBoolean:=True
MyReal:=MyBoolean
```

文字列の長さは変更できません。

```
C_STRING(3;MyString)
C_STRING(5;MyString)
```

配列の次元数は変更できません。

```
ARRAY TEXT(MyArray;5;5)
ARRAY TEXT(MyArray;5)
```

フォームの配列変数タイプが一致しません。

```
ARRAY INTEGER(MyArray)
```

配列定義コマンドに次元数がありません。

```
ARRAY INTEGER(MyArray)
```

変数が必要です。

```
COPY ARRAY(MyArray;"")
```

定数が必要です。

```
C_STRING(Variable;MyString)
```

変数タイプが不明です。この変数はメソッド"M1"で使用されています。
変数タイプが決められません。コンパイラコマンドが必要です。

定数タイプが無効です。

```
OK:="The weather is nice"
```

メソッド"M1"が不明です。
この行は存在しないメソッド、もしくは存在しなくなったメソッドを呼び出しています。

誤ったフィールドの使用。

```
MyDate:=Add to date(BooleanField;1;1;1)
```

文字列の長さは255文字（バイト）までです。

```
C_STRING(325;MyString)
```

変数Variableはメソッドではありません。

```
Variable(1)
```

変数Variableは配列ではありません。

```
Variable{5}:=12
```

結果が式と一致しません。

```
Text:="Number"+Num(i)
```

変数タイプが不適切です。

```
Integer:=MyDate*Text
```

変数$iのタイプを固定長文字列から実数に変更できません。

```
$i:="3"
${$i}:=5
```

配列のインデックスが数値ではありません。

```
IntArray{"3"}:=4
```

変数Variableのタイプをテキストから配列に変更できません。

```
C_TEXT(Variable)
COPY ARRAY(TextArray;Variable)
```

変数Variableのタイプをテキストから実数に変更できません」

```
Variable:=Num(Variable)
```

MyBooleanのタイプをブール配列から実数タイプの変数に変更できません。

```
Variable:=MyBoolean
```

配列IntArrayのタイプを整数タイプの配列からテキストタイプの配列に変更できません。

```
ARRAY TEXT(IntArray;12)
```

"IntArray"が他の所で整数配列として定義されている場合。

ポインタタイプ以外の変数をポインタとして参照しています。
Variable->:=5※"Variable"のタイプがポインタではない場合。

変数Var1のタイプをテキストから数値に変更できません。

```
Var1:=3.5
```

フィールドの使い方に誤りがあります。

```
Variable:=[MyTable]MyField
```

※[MyTable]MyFieldは日付フィールドで"Variable"は数値の場合。

シンタックス

ポインタタイプ以外の変数をポインタとして参照しています。

```
Variable:=Num("The weather is nice")->
```

この関数を参照することは使用できません。

シンタックスエラー

```
If(Boolean)
End for
```

"}"がありません。
※文中の右カッコ"}"の数が左カッコ"{"の数より少ない場合。

"{"がありません。
※文中の左カッコ"{"の数が右カッコ"}"の数より少ない場合。

")"がありません。
※文中の右カッコ")"の数が左カッコ"("の数より少ない場合。

"("形のカッコがありません。
※文中の左カッコ"("の数が右カッコ")"の数より少ない場合。

フィールドが必要です。

```
If(Modified(Variable))
```

")"がありません。

```
C_INTEGER($
```

変数が必要です。

```
C_INTEGER([MyTable]MyField)
```

定数値が必要です。

```
C_INTEGER(${"3"})
```

セミコロン(";")が必要です。

```
COPY ARRAY(Array1 Array2)
```

文字参照記号"]]"が足りません。

```
MyString[[3:="a"
```

文字参照記号"[["が足りません。

```
MyString3]]:="a"
```

サブテーブルは使用できません。

```
ARRAY TO SELECTION(Array;Subtable)
```

IF文には、ブール型が必要です。

```
If(Pointer)
```

式が複雑すぎます。
命令文を短く分割してください。

メソッドが複雑すぎます。
"Case of...End"または"If...End if"構造が多すぎます。

フィールドが不明です。
使用メソッドがおそらく他のデータベースからコピーされたもので、存在しないフィールドへの参照が式に含まれています。

テーブルが不明です。
使用メソッドがおそらく他のデータベースからコピーされたもので、存在しないテーブルへの参照が式に含まれています。

ポインタが参照する式に誤りがあります。

```
Pointer:=->Variable+3
```

文字列インデックスの使い方に誤りがあります。

```
MyReal[[3]]
```

または

```
MyString[[Variable]]
```

※"Variable"が数値変数以外の場合。

引数

この式の結果を引数としてこのメソッドもしくはコマンドに渡すことはできません。

```
MyMethod(Num(MyString))
```

※MyMethodの引数にブール式が必要な場合。

このメソッドに渡す引数が多すぎます。

```
DEFAULT TABLE(Table;Form)
```

この値を引数としてこのメソッドもしくはコマンドに渡すことはできません。

```
MyMethod(3+2)
```

※MyMethodの引数にブール式が必要な場合。

結果のタイプが一致しません。

```
C_INTEGER($0)
$0:=False
```

変数のタイプが一致しません。

```
C_INTEGER(${3})
For($i;3;5)
   ${$i}:=String($i)
End for
```

このコマンドに引数は不要です。

```
SHOW TOOL BAR(MyVar)
```

このコマンドには1つ以上の引数が必要です。

```
DEFAULT TABLE
```

MyStringを引数としてメソッドに代入することはできません。

```
MyMethod(MyString)
```

※MyMethodの引数にブール式が必要な場合。

引数"$1"のタイプが呼ぶ側のメソッドと呼ばれる側のメソッドで一致していません。

```
Calculate("3+2")
```

※メソッド"Calculate"の中でコンパイラ指示子C_INTEGER($1)が記述されている場合。

COPY ARRAYコマンドの引数に変数が含まれています。

```
COPY ARRAY(Variable;Array)
```

変数$1のタイプを数値からテキストに変更できません。

```
$1:=String($1)
```

配列を引数として使用できません。
ReInit(MyArray)配列をメソッドに代入する場合は、配列にポインタを渡してください。

演算子

演算と変数タイプが一致しません。

```
Bool2:=Bool1+True
```

ブール値のフィールドを加算することはできません。

演算子>は不要です。

```
QUERY(MyTable;[MyTable]MyField=0;>)
```

これらの変数タイプを比較することはできません。

```
If(Number=Picture2)
```

この変数タイプに-(マイナス)符号を付けることはできません。

```
Boolean:=-False
```

プラグインコマンド

プラグインコマンド"PExt"の定義が正しくありません。

プラグインコマンドに対する引数が足りません。

プラグインコマンドに対する引数が多すぎます」。

プラグインコマンドの変数の定義が正しくありません。

全般のエラー

同じ名前のメソッドが複数あります。
データベースをコンパイルするには、プロジェクトメソッドすべてに異なる名前をつける必要があります。

内部エラーNo.xx
このメッセージが出力されたら4D社テクニカルサポートに問い合わせ、エラー番号を教えてください。

変数Variableのタイプが不明です。この変数はM1メソッドで使用されています。
変数のタイプが判断できません。ココンパイラ指示子を使用してください。

オリジナルのメソッドが壊れています。
オリジナルのストラクチャでメソッドが壊れています。該当するメソッドを削除するか置き換えてください。

4Dのコマンドでありません。
このメソッドは壊れています。

フォームに配置した変数のタイプは変更できません。
フォーム内のグラフタイプの変数に"OK"といった名前をつけると、このメッセージが出力されます。

関数と変数が同じ名前です：Name
関数か変数いずれかの名前を変えてください。

メソッドと変数が同じ名前です：Name
メソッドか変数いずれかの名前を変えてください。

プラグインコマンドと変数が同じ名前です：Name
プラグインコマンドか変数いずれかの名前を変えてください。

範囲チェックメッセージ

これらのメッセージはコンパイルされたデータベースの実行中に、4Dにより表示されるエラーメッセージです。

結果が変数の範囲を越えました。

```
MyArray{17}:=2.3
```

MyArrayが要素数5つの配列であるとき、上記のステートメントが実行された場合、このメッセージはMyArray{17}にアクセスしようとすると表示されます。

ゼロによる割算が発生しました。

```
Var1:=0
Var2:=2
Var3:=Var2/Var1
```

引数がありません。
カレントプロシージャには引数が3つしか渡されていないのに、ローカル変数"$4"を使用しているような場合にこのメッセージが表示されます。

ポインタが初期化されていません。

```
MyPointer->:=5
```

i※MyPointerが初期化されていない場合。

代入先が小さすぎます。

```
C_STRING(MyString1;5)
C_STRING(MyString2;10)
MyString2:="Flowers"
MyString1:=MyString2
```

文字参照エラー。

```
i:=-30
MyString[[i]]:=MyString2
```

引数に渡された文字列が空です。

```
MyString[[1]]:=""
```

ゼロによる割算が発生しました。

```
Var1:=0
Var2:=2
Var3:=Var2% Var1
```

EXECUTE内での無効な引数。

```
EXECUTE("MyMethod(MyAlpha)")
```

※MyMethodが英数字以外の引数が必要な場合。

変数へのポインタがコンパイラには未知のものです。

```
MyPointer:=Get pointer("Variable")
MyPointer:="MyString"
```

※Variable変数がデータベース内で明確に宣言されていない場合。

ポインタを使用して再タイプ設定を試行しました。

```
Boolean:=Pointer->
```

※ポインタが整数タイプのフィールドを指す場合。

ポインタの使い方に誤りがあるか、ポインタが未知のものです。

```
Character:=StringVar[[Pointer->]]
Character:=StringVar[[Pointer]]
```

※Pointerがポインタでなく数値の場合。

C_BLOB

C_BLOB ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	→	メソッド名 (オプション)
variable	変数	→	宣言する変数名

説明

C_BLOBは、指定されたそれぞれの変数をBLOB変数としてキャストします。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースでも使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数（$0、$1、$2等）をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は"COMPILER"で開始する必要があります。

上級ヒント: シンタックスC_BLOB(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_BLOB(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

C_BOOLEAN

C_BOOLEAN ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	→	メソッド名 (オプション)
variable	変数	→	宣言する変数名

説明

C_BOOLEANは、指定されたそれぞれの変数をブール変数としてキャストします。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースでも使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数（$0、$1、$2等）をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は"COMPILER"で開始する必要があります。

上級ヒント: シンタックスC_BOOLEAN(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_BOOLEAN(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

C_DATE

C_DATE ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	⇒	メソッド名 (オプション)
variable	変数	⇒	宣言する変数名

説明

C_DATEは、指定されたそれぞれの変数を日付変数としてキャストします。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースでも使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数（$0、$1、$2等）をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は"COMPILER"で開始する必要があります。

上級ヒント: シンタックスC_DATE(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_DATE(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

C_GRAPH

C_GRAPH ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	文字	⇒	メソッド名 (オプション)
variable	変数	⇒	宣言する変数名

説明

C_GRAPHは、指定されたそれぞれの変数をグラフ変数としてキャストします。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースでも使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数（$0、$1、$2等）をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は"COMPILER"で開始する必要があります。

上級ヒント: シンタックスC_GRAPH(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_GRAPH(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

C_LONGINT

C_LONGINT ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	⇨	メソッド名 (オプション)
variable	変数	⇨	宣言する変数名

説明

C_LONGINTは、指定されたそれぞれの変数を倍長整数変数としてキャストします。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースでも使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数（$0、$1、$2等）をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は"COMPILER"で開始する必要があります。

上級ヒント: シンタックスC_LONGINT(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_LONGINT(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

C_OBJECT

C_OBJECT ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	→	メソッド名
variable	変数	→	宣言する変数の名前、または引数${...}

説明

C_OBJECT は、指定されたそれぞれの変数をランゲージオブジェクト型としてタイプ定義します。

Object 型はv14より4Dランゲージでサポートされるようになりました。この型のオブジェクトは オブジェクト(ランゲージ) テーマのコマンドで管理されます。

プロセス、インタープロセスまたはローカル変数を宣言・タイプするためには(method 引数が渡されない)第一記法を使用します。この記法はインタープリテッドデータベースにて使用可能です。

メソッドの結果と引数 ($0, $1, $2, 等)をコンパイラーへと先に宣言するためには(method 引数が渡される)第二記法を使用します。データベースがコンパイルされる際に変数タイピングフェーズを省略して時間を節約したいときにはこの記法を使用しなければなりません。

警告: 第二記法はインタープリタモードで実行することはできません。このため、この記法を使用する際には、この記法をインタープリタモードでは実行されないメソッド(名前が"COMPILER"で始まる必要があります)に保存する必要があります。

特殊使用法: C_OBJECT(${...}) の記法を使用すると、メソッドの最後の引数が同じ型であればメソッドに対して同じ型の不定数の引数を指定することができます。例えば、**C_OBJECT**(${5}) は5番目の引数以降、メソッドはこの型の引数を不定数受け取ることができるということをコンパイラに意味します。

例題

コンパイラコマンド セクションを参照してください。

C_PICTURE

C_PICTURE ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	⇒	メソッド名 (オプション)
variable	変数	⇒	宣言する変数名

説明

C_PICTUREは、指定されたそれぞれの変数をピクチャ変数としてキャストします。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースでも使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数($0、$1、$2等)をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は"COMPILER"で開始する必要があります。

上級ヒント: シンタックスC_PICTURE(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_PICTURE(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

C_POINTER

C_POINTER ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	⇒	メソッド名 (オプション)
variable		⇒	宣言する変数名

説明

C_POINTERは、指定されたそれぞれの変数をポインタ変数としてキャストします。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースでも使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数（$0、$1、$2等）をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は“COMPILER”で開始する必要があります。

上級ヒント: シンタックスC_POINTER(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_POINTER(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

C_REAL

C_REAL ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	⇒	メソッド名 (オプション)
variable	変数	⇒	宣言する変数名

説明

C_REALは、指定されたそれぞれの変数を実数変数としてキャストします。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースでも使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数（$0、$1、$2等）をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は"COMPILER"で開始する必要があります。

上級ヒント: シンタックスC_REAL(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_REAL(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

C_TEXT

C_TEXT ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	⇨	メソッド名 (オプション)
variable	変数	⇨	宣言する変数名

説明

C_TEXTは、指定されたそれぞれの変数をテキスト変数としてキャストします。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースでも使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数($0、$1、$2等)をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は"COMPILER"で開始する必要があります。

上級ヒント: シンタックスC_TEXT(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_TEXT(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

C_TIME

C_TIME ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	⇒	メソッド名 (オプション)
variable	変数	⇒	宣言する変数名

説明

C_TIMEは、指定されたそれぞれの変数を時間変数としてキャストします。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースでも使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数（$0、$1、$2等）をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は"COMPILER"で開始する必要があります。

上級ヒント: シンタックスC_TIME(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_TIME(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

IDLE

IDLE
このコマンドは引数を必要としません

説明

IDLEコマンドは、コンパイラと一緒に使用する目的だけに作成されたコマンドです。このコマンドは、4Dエンジンに呼び出しが戻らないように書かれたメソッド中で、コンパイルされたデータベースにおいてのみ使用されます。例えば、ループ内にまったく4Dコマンドを含まないForループを持ったメソッドを実行している場合、ON EVENT CALLでインストールされた割り込みメソッドでそれを中断することはできませんし、ユーザが他のアプリケーションに切り替えることもできません。このような場合、IDLEを挿入して、4Dがイベントをトラップできるようにします。割り込みを起こしたくない場合は、IDLEコマンドを記述しないでください。

例題

以下の例は、IDLEを使用しないと、コンパイルしたデータベースでループから抜け出すことができません:

```
 ` Do Something Project Method
ON EVENT CALL("EVENT METHOD")
<>vbWeStop:=False
MESSAGE("Processing..."+Char(13)+"Type any key to interrupt...")
REPEAT
 ` 4Dコマンドを呼び出さない何らかの処理を行う
   IDLE
Until(<>vbWeStop)
ON EVENT CALL("")
```

EVENT METHODは以下のとおりです:

```
 ` EVENT METHOD Project Method
If(Undefined(KeyCode))
   KeyCode:=0
End if
If(KeyCode#0)
   CONFIRM("Do you really want to stop this operation?")
   If(OK=1)
      <>vbWeStop:=True
   End if
End if
```

C_INTEGER

C_INTEGER ({method ;} variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	⇒	メソッド名 (オプション)
variable		⇒	宣言する変数名

予備的な注意

このコマンドは古いデータベースとの互換性のために存在しています。4Dは整数を倍長整数に型変更します。例えば：

```
C_INTEGER($MyVar)
$TheType:=Type($MyVar) `$TheType = 9 (Is Longint)
```

説明

C_INTEGERは、指定されたそれぞれの変数を整数変数としてキャストします。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースで使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数（$0、$1、$2等）をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は"COMPILER"で開始する必要があります。

上級ヒント: シンタックスC_INTEGER(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_INTEGER(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

C_STRING

C_STRING ({method ;} size ; variable {; variable2 ; ... ; variableN})

引数	型		説明
method	メソッド	⇨	Nom de méthode
size	倍長整数	⇨	文字列のサイズ
variable	変数	⇨	宣言する変数名

説明

C_STRINGは、指定されたそれぞれの変数を文字列変数としてキャストします。

size 引数は、変数が格納することのできる最大文字長を指定します。非Unicodeモード (ASCII互換) では、文字列は255文字に制限されます。速度を考慮する場合、可能な限りテキスト変数よりも文字列変数を使用します。

互換性に関する注意: 4Dのバージョン11以降で作成されたデータベースは、デフォルトでUnicodeモードで実行されます (の節参照)。このモードでは、C_STRING コマンドの動作はC_TEXT コマンドとまったく同じです (size 引数は無視されます)。新規の開発ではC_TEXTの利用をお勧めします。C_STRING コマンドは互換性のために保持されています。

コマンドの第1の形式は、オプションのmethod引数が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプ定義に使用されます。

Note: この形式はインタプリタのデータベースで使用できます。

コマンドの第2の形式は、オプションのmethod引数が渡される形式であり、メソッドの結果や引数 ($0、$1、$2等) をコンパイラ用に事前定義するために使用されます。このコマンドの形式は、データベースのコンパイル中に、変数設定フェーズをスキップし、コンパイル時間を節約するために使用します。

警告: 2番目の形式をインタープリタモードで実行できません。このため、このシンタックスは、インタープリタモードで実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は"COMPILER"で開始する必要があります。

上級ヒント: シンタックスC_STRING(${...})を使用すると、同じ型の不定数の引数を宣言できます。ただしこれらの引数はメソッドの最後の引数でなければなりません。例えば、C_STRING(${5})宣言は、4Dとコンパイラに対して、5番目のパラメータ以降、そのメソッドがそのタイプの不定数の引数を受け付けることを伝えています。詳細は、Count parametersコマンドを参照してください。

例題

の節を参照

サブレコード

- Get subrecord key
- ALL SUBRECORDS
- APPLY TO SUBSELECTION
- Before subselection
- CREATE SUBRECORD
- DELETE SUBRECORD
- End subselection
- FIRST SUBRECORD
- LAST SUBRECORD
- NEXT SUBRECORD
- ORDER SUBRECORDS BY
- PREVIOUS SUBRECORD
- QUERY SUBRECORDS
- Records in subselection

Get subrecord key

Get subrecord key (idField) -> 戻り値

引数	型		説明
idField	フィールド	→	以前のサブテーブルリレーションの"サブテーブルリレーション"または"倍長整数"型のフィールド
戻り値	倍長整数	↩	リレーションの内部キー

説明

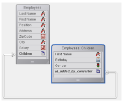
Get subrecord keyコマンドは、変換されたサブテーブルを使用する4Dコードから、標準のテーブルに対して作業を行うコードへの移行を容易にします。

注意: 4Dバージョン11よりサブテーブルはサポートされていません。旧バージョンのデータベースを変換すると、既存のサブテーブルは標準のテーブルに変換され、自動リレーションにより親テーブルとリンクされます。以前のサブテーブルはNテーブルとなり、親テーブルは1テーブルになります。1テーブル中、以前のサブテーブルフィールドは"サブテーブルリレーション"型の特別なフィールドに変換され、Nテーブルには特別な"サブテーブルリレーション"型のフィールドが"id_added_by_converter"という名称で追加されます。

これは変換されたデータベースにおいてサブテーブルの動作の互換性を保持するためのものです。しかし変換されたデータベースにおいて、すべてのサブテーブルメカニズムを標準のテーブルを使用したメカニズムに置き換えることを強く推奨します。

この作業ではまず特別な自動リレーションを削除してサブテーブルから継承したメカニズムを恒久的に無効にし、その後関連するコードを書き換える必要があります。**Get subrecord key**コマンドはリレーションで使用される内部IDを返し、このIDを使用することでこの書き換えを容易に行うことができます。この内部IDは実際のリレーションを不要とし、リレーションがもう存在しないにもかかわらず、開発者は以前のサブテーブルのセレクションで作業を行うことができます。

変換された以下のストラクチャーで例題を見てみましょう:



4D v12では以下のコードが依然動作しますが、更新する必要があります:

```
ALL SUBRECORDS([Employees]Children)
$total:=Records in subselection([Employees]Children)
vFirstnames:=""
For($i;1;$total)
   vFirstnames:=vFirstnames+[Employees]Children'FirstName+" "
   NEXT SUBRECORD([Employees]Children)
End for
```

このコードを以下のように書き換えることができます:

```
QUERY([Employees_Children];[Employees_Children]id_added_by_converter=Get subrecord key([Employees]Children))
$total:=Records in selection([Employees_Children])
vFirstnames:=""
For($i;1;$total)
   vFirstnames:=vFirstnames+[Employees_Children]FirstName+" "
   NEXT RECORD(Employees_Children)
End for
```

注: Get subrecord keyコマンド実行時、カレントレコードがロードされていない場合は0を返します。

2番目のコードは標準の4Dコマンドを使用していますが、リレーションが存在するしないにかかわらず以前と同様に動作します。リレーションを取り除くと、コマンドは倍長整数フィールドに格納された値を返します。

idField引数にはサブテーブルリレーション型のフィールド (リレーションがまだ存在する場合) または倍長整数タイプのフィールド (リレーションを取り除いた場合) を渡します。これ以外の場合エラーが生成されます。

このコマンドを使用することで、遷移的なコードを書くことができます。アプリケーションのアップグレードの最終ステージで、このコマンドの呼び出しを取り除くことができます。

ALL SUBRECORDS

ALL SUBRECORDS (subtable)

引数	型		説明
subtable	サブテーブル	→	すべてのサブレコードを選択するサブテーブル

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

ALL SUBRECORDSはsubtableのすべてのサブレコードをカレントサブセレクションにします。現在の親レコードが存在しない場合、ALL SUBRECORDSは機能しません。最初に親のレコードがロードされると、サブセレクションはすべてのサブレコードを格納します。ADD SUBRECORD、QUERY SUBRECORDS、またはDELETE SUBRECORDを実行すると、サブセレクションはすべてのサブレコードを格納しないこともあります。

例題

以下の例で、すべてのサブレコードを選択し、それらを確実に合計に取り入れます。

```
ALL SUBRECORDS([Stats]Sales)
TotalSales:=Sum([Stats]Sales'Dollars)
```

APPLY TO SUBSELECTION

APPLY TO SUBSELECTION (subtable ; statement)

引数	型		説明
subtable	サブテーブル	⇒	フォーミュラを適用するサブテーブル
statement	命令文	⇒	コードまたはメソッド

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

APPLY TO SUBSELECTIONは、statementをsubtableのカレントサブセレクションにある各サブレコードに適用します。statementはステートメント、またはメソッドのいずれかです。statementがサブレコードを修正する場合、親のレコードが書き込まれたときのみ、修正されたサブレコードはディスクに書き込まれます。サブセレクションが空の場合、APPLY TO SUBSELECTIONは機能しません。

APPLY TO SUBSELECTIONを使用して、サブセレクションから情報を収集したり、サブセレクションを修正したりすることができます。

Before subselection

Before subselection (subtable) -> 戻り値

引数	型		説明
subtable	サブテーブル	→	サブレコードポインタがサブレコードセレクションの 前にあるかテストするサブテーブル
戻り値	ブール	←	Yes (TRUE)またはNo (FALSE)

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

カレントサブレコードのポインタがsubtableの最初のサブレコードの前にある場合、Before subselection はTrueを返します。Before subselectionを使用して、PREVIOUS SUBRECORDがポインタを最初のサブレコードの前に移動させたかどうかを確認します。カレントサブセレクションが空の場合、Before subselectionはTrueを返します。

例題

以下は、ボタンのオブジェクトメソッドの使用例です。ボタンをクリックすると、ポインタは前のサブレコードへ移動します。ポインタが最初のサブレコードの前にある場合、ポインタは最後のサブレコードへ移動します。

```
PREVIOUS SUBRECORD([People]Children) `前のサブレコードへ移動
If(Before subselection([People]Children) `行き過ぎたら...
   LAST SUBRECORD([People]Children) `最後のサブレコードへ移動
End if
```

CREATE SUBRECORD

CREATE SUBRECORD (subtable)

引数	型		説明
subtable	サブテーブル	→	新しいサブレコードを作成するためのサブテーブル

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

CREATE SUBRECORDは、subtable用の新しいサブレコードを作成し、その新しいサブレコードをカレントサブレコードにします。その親のレコードが保存された時のみ、新しいサブレコードは保存されます。親のレコードはSAVE RECORDなどのコマンドやユーザがレコードを受け入れることによって保存されます。カレントレコードがない場合、CREATE SUBRECORDはその効力を発揮しません。サブレコード入力フォームを通じて新しいサブレコードを追加するには、ADD SUBRECORDを使用します。

例題

ボタンのオブジェクトメソッドを以下の例に表します。このメソッドが実行された場合(ボタンがクリックされた時) 、[People] テーブルにある子どものレコードにサブレコードを作成します。ユーザは、Repeatループで子どもを追加できます。キャンセルボタンをクリックすると中止します。サブフォームにある子どもがフォームで表示されますが、入力可オプションがチェックされていないので、直接サブテーブルにデータを入力することはできません。

```
Repeat
  ` Get the child's name
   vChild:=Request("Name (終了したらキャンセル) :")
  `ユーザがOKをクリックすると
   If(OK=1)
  `子どもの新しいサブレコードを追加します
      CREATE SUBRECORD([People]Children)
  `サブフィールドへ子どもの名前を割り当てる
      [People]Children'Name:=vChild
   End if
Until(OK=0)
```

DELETE SUBRECORD

DELETE SUBRECORD (subtable)

引数	型	説明
subtable	サブテーブル	→ カレントサブレコードを削除するサブテーブル

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

DELETE SUBRECORDはsubtableのカレントサブレコードを削除します。カレントサブレコードがない場合、DELETE SUBRECORDはその効力を発揮しません。サブレコードが削除されると、subtableのカレントサブセレクションは空になります。従って、サブセレクションをスキャンして選択したサブレコードを削除する用途でDELETE SUBRECORDを使用することはできません。

親のレコードが保存されるまで、サブレコードの削除は永続的ではありません。親のレコードを削除すると、すべてのサブレコードは自動的に削除されます。

サブセレクションを削除するには、削除したいサブセレクションを作成し、最初のサブレコードを削除します。再度、サブセレクションを作成し、最初のサブレコードを削除します。この作業を繰り返します。

例題 1

一つのサブテーブルのすべてのサブレコードを削除する例を以下に表します。

```
ALL SUBRECORDS([People]Children)
While(Records in subselection([People]Children)>0)
   DELETE SUBRECORD([People]Children)
   ALL SUBRECORDS([People]Children)
End while
```

例題 2

子どもの年齢が12歳以上のサブレコードを[People]Childrenサブテーブルから削除する例を以下に表します。

```
ALL RECORDS([People])  `すべてのレコードを選択します。
For($vlRecord;1;Records in selection([People]))  `テーブルのすべてのレコード用
 `条件付きサブレコードを持つすべてのレコードをクエリする
   QUERY SUBRECORDS([People]Children;[People]Children'Age>=12)
 `すべてのサブレコードがなくなるまでクエリを繰り返す
   While(Records in subselection([People]Children)>0)
 `サブレコードを削除する
      DELETE SUBRECORD([People]Children)
 `再度クエリする
      QUERY SUBRECORDS([People]Children;[People]Children'Age>=12)
   End while
   SAVE RECORD([People])  `親のレコードを保存する
   NEXT RECORD([People])
End for
```

End subselection

End subselection (subtable) -> 戻り値

引数	型		説明
subtable	サブテーブル	→	サブレコードのポインタがサブレコードセレクション より後にあるかをテストするサブテーブル
戻り値	ブール	↩	Yes (TRUE)またはNo (FALSE)

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

End subselection はカレントサブレコードポインタがsubtableのカレントサブセレクションの終わりより後ろにあればTrueを返します。End subselection は、NEXT SUBRECORD がポインタを最後のサブレコードの後ろに移動させたかをチェックするために使用します。カレントサブセレクションが空の場合、End subselection はTrueを返します。

例題

以下は、ボタンのオブジェクトメソッドの使用例です。ボタンをクリックすると、ポインタは次のサブレコードへ移動します。ポインタが最後のサブレコードの後にある場合、ポインタは最初のサブレコードへ移動します。

```
NEXT SUBRECORD([People]Children) `次のサブレコードへ移動
If(End subselection([People]Children)) `行き過ぎたら...
   FIRST SUBRECORD([People]Children) `最初のサブレコードへ移動
End if
```

FIRST SUBRECORD

FIRST SUBRECORD (subtable)

引数	型		説明
subtable	サブテーブル	→	最初に選択されたサブレコードへ 移動するサブテーブル

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

FIRST SUBRECORD はsubtable のカレントサブセレクションの最初のサブレコードをカレントサブレコードにします。すべてのクエリ、セレクション、並び替えコマンドも最初のサブレコードをカレントのサブレコードにします。カレントサブセレクションが空の場合FIRST SUBRECORD はなにも行いません。

例題

次の例を使用して、サブテーブルに格納されている子どものレコードにある苗字と名前を結び付けます。そして、その氏名を配列 atNamesにコピーします。

```
  ` 名前を保持する配列を作成する
 ARRAY TEXT(atNames;Records in subselection([People]Children))
 FIRST SUBRECORD([People]Children)  ` 最初のサブレコードで開始し、各子どもに対して一度ループする
 For($vlSub;1;Records in subselection([People]Children))
    atNames{$vlSub}:=[People]Children'First Name+" "+[People]Children'Last Name
    NEXT SUBRECORD([People]Children)
 End for
```

LAST SUBRECORD

LAST SUBRECORD (subtable)

引数	型		説明
subtable	サブテーブル	→	最後に選択されたサブレコードへ 移動するサブテーブル

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

LAST SUBRECORD はsubtable のカレントサブセレクションの最後のサブレコードをカレントサブレコードにします。カレントサブセレクションが空の場合LAST SUBRECORD はなにも行いません。

例題

次の例を使用して、サブテーブルに格納されている子どものレコードにある姓と名を結び付けます。そして、その氏名を配列 atNamesにコピーします。サブレコードが最初から最後まで移動することを除いては、FIRST SUBRECORD の例と同じです。

```
  ` 名前を保持する配列を作成する
 ARRAY TEXT(atNames;Records in subselection([People]Children))
 LAST SUBRECORD([People]Children)  ` 最後のサブレコードで開始し、各子どもに対して一度ループする
 For($vlSub;1;Records in subselection([People]Children))
    atNames{$vlSub}:=[People]Children First Name+" "+[People]Children Last Name
    PREVIOUS SUBRECORD([People]Children)
 End for
```

NEXT SUBRECORD

NEXT SUBRECORD (subtable)

引数	型	説明
subtable	サブテーブル	➡ 次に選択されたサブレコードをへ 移動するサブテーブル

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

NEXT SUBRECORD はsubtable のカレントサブセレクション中でサブレコードポインタを次のサブレコードに移動します。NEXT SUBRECORD が最後のサブレコードを超えてポインタを移動したら、End subselection がTRUEを返し、カレントサブレコードはなくなります。End subselection がTRUEを返したら、FIRST SUBRECORDまたはLAST SUBRECORDを使用してカレントサブセレクション中にポインタを移動します。カレントサブセレクションが空の場合、またはBefore subselectionがTrueを返す場合、NEXT SUBRECORD はなにも行いません。

例題

FIRST SUBRECORDの例を参照してください。

ORDER SUBRECORDS BY

ORDER SUBRECORDS BY (subtable ; subfield {; > or <}{; subfield2 ; > or <2 ; ... ; subfieldN ; > or <N})

引数	型		説明
subtable	サブテーブル	→	選択されたサブレコードを並べるためのサブテーブル
subfield	サブフィールド	→	各レベルごとにサブフィールドで並べる
> or <	演算子	→	各レベルに方向を指示し、> 昇順で並べ替え、または < 降順で並べ替え

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

ORDER SUBRECORDS BYは、subtableのカレントサブセレクションをソートします。カレントの親レコードに属するサブテーブルのサブセレクションのみをソートします。

引数directionは、subfieldを昇順または降順でソートするかを指定します。direction が"大なり" 記号 (>) の場合、サブレコードの並べ替えは昇順となります。directionが"小なり" 記号 (<) の場合、サブレコードの並べ替えは降順となります。

複数のサブフィールドと並び替え方向を定義して、ソートのレベルを1つ以上指定できます。

ソートが完了すると、ソートされたサブセレクションの最初のサブレコードは、カレントサブレコードになります。サブレコードのソートは一時的な処理 です。ソートされた順番でサブレコードを格納することはありません。カレントレコード、または上位レベルのサブレコードが存在しない場合、ORDER SUBRECORDS BYは機能しません。

フォームがサブフォームを含み、そのサブフォームが固定のフレームで印刷される場合は、親フォームのフォームメソッドのBeforeフェーズで印刷を実行する前に、一度、このコマンドを呼ぶ必要があります。

例題

以下の例を使用して、SalesDollarsサブフィールドを基に、[Stats]Salesサブテーブルを昇順に並べ替えます。

```
ORDER SUBRECORDS BY([Stats]Sales;[Stats]Sales'Dollars;>)
```

PREVIOUS SUBRECORD

PREVIOUS SUBRECORD (subtable)

引数	型		説明
subtable	サブテーブル	→	前の選択されたサブレコードへ 移動させるサブテーブル

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

PREVIOUS SUBRECORD はsubtable のカレントサブセレクション中でサブレコードポインタを前のサブレコードに移動します。PREVIOUS SUBRECORDが最初のサブレコードの前にポインタを移動したら、Before subselection がTRUEを返し、カレントサブレコードはなくなります。Before subselection がTRUEを返したら、FIRST SUBRECORDまたはLAST SUBRECORDを使用してカレントサブセレクション中にポインタを移動します。カレントサブセレクションが空の場合、またはEnd subselectionがTrueを返す場合、PREVIOUS SUBRECORDはなにも行いません。

例題

LAST SUBRECORDの例を参照してください。

QUERY SUBRECORDS

QUERY SUBRECORDS (subtable ; queryFormula)

引数	型		説明
subtable	サブテーブル	⇒	検索するサブテーブル
queryFormula	ブール	⇒	クエリフォーミュラ

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

QUERY SUBRECORDSはsubtableをクエリし、新しいサブセレクションを作成します。これはサブレコードをクエリし、サブレコードのセレクションを返す唯一のコマンドです。queryFormulaはsubtableにある各サブレコードに適用されます。フォーミュラがTRUEとして評価される場合、サブレコードは新しいセレクションへ追加されます。クエリが完成すると、QUERY SUBRECORDS は最初のサブレコードをsubtableのカレントサブレコードにします。

QUERY SUBRECORDSは、その時点で選択されている親のレコードに含まれるサブテーブルのサブレコードだけをクエリします。親テーブルに属するすべてのレコードクエリするわけではありません。QUERY SUBRECORDSは、親のカレントレコードを変更しません。

典型的な例として、queryFormulaは関係演算子を使用して、変数または定数に対してサブフィールドをテストします。queryFormulaに複数のテストを格納し、AND (&) または OR (|) で結合することができます。またqueryFormulaは関数であったり、関数を含んでいたりします。ワイルドカード文字 (@) を文字引数で使用できます。

カレントレコードまたは上位レベルのサブレコードが存在しない場合、QUERY SUBRECORDSは機能しません。

例題

以下の例を使用して、10歳を超える子どもをクエリします。

```
QUERY SUBRECORDS([People]Children;[People]Children'Age>10)
```

Records in subselection

Records in subselection (subtable) -> 戻り値

引数	型		説明
subtable	サブテーブル	→	サブレコードの数を数えるためのサブテーブル
戻り値	倍長整数	↩	カレントサブセレクションにあるサブレコードの数

互換性に関するメモ

バージョン11以降の4Dはサブテーブルをサポートしていません。互換性メカニズムは、変換されたデータベースでコマンドの機能を保護しますが、すべてのサブテーブルをリレートする標準的なテーブルに取り換えることを強くお勧めします。

説明

Records in subselectionはsubtableのカレントサブセレクションにあるサブレコードの数を返します。Records in subselectionはカレントレコードにあるサブレコードのみに適用されます。このコマンドはサブレコード用のRecords in selectionに相当します。親のレコードが存在しない場合、その結果は定義されません。

例題

以下の例を使用して、すべてのサブレコードを選択し、親のレコードに対する子どもの数を表示します。

```
  `すべての子どもを選択し、その総数を表示
 ALL SUBRECORDS([People]Children)
 ALERT("Number of children: "+String(Records in subselection([People]Children)))
```

システムドキュメント

- システムドキュメント
- Append document
- CLOSE DOCUMENT
- Convert path POSIX to system
- Convert path system to POSIX
- COPY DOCUMENT Updated 14.0
- CREATE ALIAS
- Create document
- CREATE FOLDER
- DELETE DOCUMENT
- DELETE FOLDER
- Document creator
- DOCUMENT LIST
- Document to text New 14.0
- FOLDER LIST
- GET DOCUMENT ICON
- Get document position
- GET DOCUMENT PROPERTIES
- Get document size
- Get localized document path
- MAP FILE TYPES
- MOVE DOCUMENT
- Open document
- RESOLVE ALIAS
- Select document
- Select folder
- SET DOCUMENT CREATOR
- SET DOCUMENT POSITION
- SET DOCUMENT PROPERTIES
- SET DOCUMENT SIZE
- SET DOCUMENT TYPE
- SHOW ON DISK
- Test path name
- TEXT TO DOCUMENT New 14.0
- VOLUME ATTRIBUTES
- VOLUME LIST
- Document type

システムドキュメント

概要

コンピューターで使用する全てのドキュメントとアプリケーションは、自身のマシンに**実装**または**接続**されているハードディスクやフロッピーディスク、その他の類似する恒久的なストレージ装置内にファイルとして格納されています。4Dでは、これらのドキュメントやアプリケーションに言及する際、**ファイル**または**ドキュメント**という表現を使用します。しかし、このテーマのほとんどのコマンドにおいては、"ドキュメント" という表現が使用されています。それは、多くの時間、ユーザはこれらのコマンドを使用して、(アプリケーションやシステムファイルよりも) ディスク内のドキュメントにアクセスするからです。

ハードディスクは、1つまたは複数のパーティションとしてフォーマットされます。それぞれのパーティションは**ボリューム**と呼ばれ、2つのボリュームが、物理的に同じハードディスク内に存在しても問題はありません。4Dの最初のレベルでは、通常、これらのボリュームを個別で等しい実体として扱います。

ボリュームはマシンに物理的に接続されているハードディスク、あるいはTCP/IPやAFP、SMBなどのようなファイル共有プロトコルを通してマウントされたハードディスクに置かれています。いかなる場合でも、4Dレベルでシステムドキュメントコマンドを使用する際、これら全てのボリュームを同じ方法で扱います (方法について熟知していて、プラグインを使用してアプリケーションの能力を拡大する場合は例外です) 。

それぞれのボリュームには、**ボリューム名** が付けられています。Windowsではボリューム名の後にコロンを続けてボリュームを指定します。通常CやDはシステムを起動するボリュームを指定するために使用されます (特にPCで設定変更しない場合) 。EからZの文字は、PC(DVDドライブ、追加ドライブ、ネットワークドライブ等) に接続、または実装されている付加ボリューム用に使用されます。Macintoshでは、ボリューム名はそのままです。これらは、デスクトップ上のFinderレベルで見ることができます。

通常、ドキュメントを**フォルダー**へ分類します。そして、そのフォルダー自体、他のフォルダーを格納しています。同じボリュームのレベルで何千というファイルを蓄積するのは良い方法ではありません。乱雑ですし、システムの速度を低下させます。Windowsでは、フォルダーを**ディレクトリ**と呼び、Macintoshでは、そのままフォルダーと呼びます。

ドキュメントを区別するには、そのドキュメントが保存されているフォルダの名前とボリュームの名前、そして、そのドキュメント自体の名前を知る必要があります。これらすべての名前を連結し、そのドキュメントへの**パス名** を取得します。このパス名においてフォルダ名は、**フォルダーセパレーター記号**と呼ばれる特殊文字で区切られます。Windowsではその文字はバックスラッシュ(\: 日本語フォント環境では円マーク)です。Macintoshでは、コロン(:)になります。

では例を見てみましょう。Current Workフォルダー内にあるDocumentsフォルダーのMemosフォルダーに保存されているドキュメントImportant Memoがあります。

Windowsでは、これらのすべてがCドライブ(ボリューム) に保存されていれば、そのドキュメントのパス名は次のようになります。

C:\Current Work\Documents\Memos\Important Memo.TXT

注: バックスラッシュ(\) は、4Dのメソッドエディタでエスケープシーケンスを表すために使用されます。解釈上の問題を回避するために、エディタは自動的にパス名を変更します。例えば、C:\DiskからC:\\Diskへと変換します。詳細については後述の"ドキュメント名、またはドキュメントのパス名を指定する" 項目を参照してください。.

Macintoshではこれらの全てが、Internal Driveディスク(ボリューム) に保存されていれば、そのドキュメントのパス名は次のようになります。

Internal Drive:Current Work:Documents:Memos:Important Memo

Windowsの例ではドキュメントの名前に接尾辞.TXTが付いています。その理由は、次のセクションで説明します。

いかなるプラットフォームでも、ドキュメントの完全なパス名は次のように表示されます。

VolName DirSep { DirName DirSep { DirName DirSep { ... } } } DocName

ボリュームにあるすべてのドキュメント (ファイル) には、幾つかの特性があります。通常、**属性**または**プロパティ**と呼ばれるものと、ドキュメントの**名前**や**タイプ**、**クリエータ**などがあります。

DocRef: ドキュメント参照番号

ドキュメントは、**読み/書きモードで開く**、**読み込み専用モードで開く**、または**閉じます**。ビルトインの4Dコマンドを使用して、一度のプロセスでドキュメントを読み/書きモードで、素早く開くことができます。1つのプロセスで複数のドキュメントを開くことが可能です。複数のプロセスでは、ドキュメントを多重に開くことができます。読み込み専用モードで同じドキュメントを必要な回数だけ開くことができます。しかし、読み/書きモードでは、一度に同じドキュメントを2つ開くことはできません。

Open document、Create documentそしてAppend documentコマンドを使用してドキュメントを開きます。Create documenとAppend documentコマンドは、自動的にドキュメントを読み/書きモードで開きます。Open documentコマンドを使用する場合のみ、開くモードを選択できます。ドキュメントが開くと、ドキュメントから読み込みをしたり、ドキュメントへ書き出しをしたりできます (RECEIVE PACKETとSEND PACKETコマンドを参照) 。ドキュメントでの作業が終了したらCLOSE DOCUMENTコマンドを使用してドキュメントを閉じます。

全ての開かれたドキュメントは、Open document、Create documentそしてAppend documentコマンドから返された**DocRef**式を使用して参照されます。DocRefは開かれたドキュメントを識別するために使用します。DocRefは時間タイプの式です。開かれたドキュメントを操作する全てのコマンドは、引数としてDocRefを受け取ります。誤ったDocRefをこれらのコマンドの1つに渡すと、ファイルマネージャエラーが発生します。

I/Oエラーを処理する

ドキュメントにアクセス (開く、閉じる、削除する、名前を変更する、コピーする) したり、ドキュメントのプロパティを変更したり、ドキュメントで文字を読んだり書いたりすると、I/Oエラーが発生することがあります。ドキュメントが見つからないこともありますし、ロックされていることもあります。また、書き込みモードで既に開いていることもあります。ON ERR CALLを用いてインストールされたエラー処理メソッドで、これらのエラーを検知することができます。システムドキュメントを使用している間に発生するほとんどのエラーに関しては、で説明しています。

ドキュメントシステム変数

コマンドOpen document、Create document、Append documentそしてSelect documentを使用して、標準的なファイルを開くまたは保存するダイアログボックス経由でドキュメントへアクセスできます。 標準ダイアログを通してドキュメントにアクセスすると、Documentシステム変数にドキュメントの完全なパス名が返されます。このシステム変数は、コマンドの引数リストに表示される引数documentと区別されなければなりません。

ドキュメント名またはドキュメントパスを指定する

この節にある、ドキュメント名の必要なほとんどのルーチンでは、ドキュメント名とパス名の両方を使用できます (特に指定された場合は例外) 。名前を渡した場合、コマンドはデータベースのフォルダ内でドキュメントを探します。パス名を渡した場合、それは有効でなければなりません。

正しくない名前やパス名を渡した場合、コマンドはファイルマネージャエラーを発生させます。このエラーはON ERR CALL メソッドを使用して検知することが可能です。

Windowsのパス名の入力とエスケープシーケンス

4Dのメソッドエディタではエスケープシーケンスの使用が可能です。エスケープシーケンスとは、文字 列のセットで、"特殊" 文字を置き換えるために使用します。シーケンスはバックスラッシュ(\: 日本語フォント環境では円マーク) で始まり、その後に文字が続きます。例えば、\t は、Tab 文字のエスケープシーケンスになります。
Windowsでは、\文字を使用して、パス名を区切ります。4Dでは、ほとんどの場合、シングルバックスラッシュをダブルバックスラッシュに置き換えることによって、Windowsのメソッドエディタに入力されているパス名を、正確に判断します。例を挙げると、C:\Folder はC:\\Folderとなります。
しかし、C:\MyDocuments\Newと書くと、4DはC:\\MyDocuments\Newと表示します。この場合、2番目の\は、\N (存在しているエスケープシーケンス) として正確に判断されません。従って、4Dで識別されているエスケープシーケンスの1つで使用される文字の前にバックスラッシュを挿入したいときは、手入力でで\\を入力しなければなりません。

4Dは次のエスケープシーケンスを識別します。

エスケープシーケンス	置き換え文字
\n	LF (New line)
\t	HT (Horizontal tab)
\r	CR (Carriage return)
\\	\ (Backslash)
\"	" (Quotes)

絶対 / 相対パス名

ドキュメントやフォルダーを管理する4Dコマンドは多くの場合相対パスと絶対パスの両方をサポートします:

- **相対パス名** はディスク上の任意のディレクトリを起点とし、そこから目的のファイルやフォルダーまでの位置関係を記述するものです。4Dの場合、通常相対パスの起点はデータベースフォルダー、つまりストラクチャーファイルが格納されているフォルダーとなります。
- **絶対パス名** はボリュームのルートを起点として場所を記述します。データベースフォルダーの所在場所は関連しません。

コマンドに渡されるパス名が相対・絶対どちらで解釈されるべきかを決定するため、4Dは各プラットフォームごとに特定のアルゴリズムを適用します。

Windows

絶対パスと解釈されるケース

- パス名が二文字だけで構成され、二文字目がコロンの場合 ':'
- パス名の二文字目と三文字目がそれぞれコロンとバックスラッシュ (円マーク) の場合
- パス名が "\\" で始まる場合

その他のケースはすべて**相対パス**と解釈されます。

CREATE FOLDERコマンドの例題:

```
CREATE FOLDER("Monday") // 相対パス
CREATE FOLDER("\Monday") // 相対パス
CREATE FOLDER("\Monday\Tuesday") // 相対パス
CREATE FOLDER("c:") // 絶対パス
CREATE FOLDER("d:\Monday") // 絶対パス
CREATE FOLDER("\\srv-Internal\temp") // 絶対パス
```

Mac OS

相対パスと解釈されるケース

- パス名がディレクトリ区切り文字 ':' で始まる場合
- パス名がディレクトリ区切り文字を含まない場合

それ以外のケースはすべて**絶対パス**です。

CREATE FOLDERコマンドの例題:

```
CREATE FOLDER("Monday") // 相対パス
CREATE FOLDER("macintosh hd:") // 絶対パス
CREATE FOLDER("Monday:Tuesday") // 絶対パス (Mondayがボリューム名)
CREATE FOLDER(":Monday:Tuesday") // 相対パス
```

ディスク上でドキュメントを処理する時に使用する便利なプロジェクトメソッド

- 起動しているプラットフォームを検知する

プラットフォームの仕様上の違いに基づく複数のコーディングを避けるために、4DはMAP FILE TYPESのようなコマンドを提供していますが、ディスク上のドキュメントを処理するような低レベルで作業を開始するときは (例えば、プログラミングでパス名を取得するなど) 、MacintoshまたはWindowsのプラットフォームのどちらで起動しているかを知る必要があります。

次のOn Windowsプロジェクトメソッドは、データベースがWindows上で起動しているかどうかをテストします。

```
  // On windows プロジェクトメソッド
  // On windows -> ブール
  // On windows -> WindowsであればTrue

 C_BOOLEAN($0)
 $0:=(Folder separator="\\")
```

- 正しいディレクトリ区切り記号を使用する

Windowsではディレクトリレベルをバックスラッシュ(\) によって表します (日本語フォント環境では円マーク)。Macintoshはフォルダレベルをコロン(:) によって表します。どちらのプラットフォームで起動しているかに基づいて、次のDirectory symbolプロジェクトメソッドは、正確なディレクトリ記号(文字) のコードを返します。

```
  // Directory symbol プロジェクトメソッド
  // Directory symbol -> 整数
  // Directory symbol -> "\" (Windows) または ":" (Mac OS)のコード

 C_LONGINT($0)

 If(On Windows)
    $0:=Character code("\\")
 Else
    $0:=Character code(":")
 End if
```

互換性に関する注意: 4Dバージョン12より、Folder separator定数 (System Documentsテーマ) を使用して、OSに左右されない有効なパス名を構築することをお勧めします。この定数は自動でOSに対応する区切り文字を返します (Unicodeモードのみ)。

- 長い名前からファイル名を抽出する

長いドキュメント名を取得してしまうと(パス名+ファイル名) 、その長い名前からドキュメントのファイル名を抽出しなければならないことがあります。例えば、ウィンドウのタイトルにその名前を表示する場合などです。Long name to file name プロジェクトメソッドは、WindowsとMacintoshの両方でこれを実行します。

```
  ` Long name to file name プロジェクトメソッド
  ` Long name to file name ( 文字列 ) -> 文字列
  ` Long name to file name ( Long file name ) -> file name

 C_STRING(255;$1;$0)
 C_INTEGER($viLen;$viPos;$viChar;$viDirSymbol)

 $viDirSymbol:=Directory symbol
 $viLen:=Length($1)
 $viPos:=0
 For($viChar;$viLen;1;-1)
    If(Character code($1[[$viChar]])=$viDirSymbol)
       $viPos:=$viChar
       $viChar:=0
    End if
 End for
 If($viPos>0)
    $0:=Substring($1;$viPos+1)
 Else
    $0:=$1
 End if
 If(<>vbDebugOn) `On Startupデータベースメソッドで、変数をTrueまたはFalseで設定する。
    If($0="")
       TRACE
    End if
 End if
```

- 長い名前からパス名を抽出する

長いドキュメント名を取得してしまうと(パス名+ファイル名) 、その長い名前からドキュメントが置かれているディレクトリのパス名を抽出しなければならないことがあります。例えば、同じ場所にドキュメントを追加して保存したい場合などです。Long name to file name プロジェクトメソッドは、WindowsとMacintoshの両方でこれを実行します。

```
  ` Long name to path name プロジェクトメソッド
  ` Long name to path name ( 文字列 ) -> 文字列
  ` Long name to path name ( Long file name ) -> Path name

 C_STRING(255;$1;$0)
 C_STRING(1;$vsDirSymbol)
 C_INTEGER($viLen;$viPos;$viChar;$viDirSymbol)

 $viDirSymbol:=Directory symbol
 $viLen:=Length($1)
 $viPos:=0
 For($viChar;$viLen;1;-1)
    If(Character code($1[[$viChar]])=$viDirSymbol)
       $viPos:=$viChar
       $viChar:=0
    End if
 End for
 If($viPos>0)
    $0:=Substring($1;1;$viPos)
 Else
    $0:=$1
 End if
 If(<>vbDebugOn) `On Startupデータベースメソッドで、変数をTrueまたはFalseで設定する。
    If($0="")
       TRACE
    End if
 End if
```

Append document

Append document (document {; fileType}) -> 戻り値

引数	型		説明
document	文字	→	ドキュメント名、または 完全なドキュメントパス名、または 標準ファイルダイアログボックスの空の文字列
fileType	文字	→	スクリーンされるドキュメントタイプのリスト、または ドキュメントをスクリーンしない場合 ""
戻り値	DocRef	↩	ドキュメント参照番号

説明

Append documentコマンドはOpen documentコマンドと同じ機能を提供します。このコマンドを使用してディスクにあるドキュメントを開くことができます。

この2つのコマンドは多少異なります。Append documentはドキュメントの最後にファイル位置を設定します。一方Open documentはドキュメントの先頭にファイル位置を設定します。

Append documentの使用についての詳細は、Open documentを参照してください。

例題

次の例を使用して、Noteという名称の既存のドキュメントを開きます。文字列"　また会いましょう"とキャリッジリターンをドキュメントの最後に付加し、ドキュメントを閉じます。例えば、ドキュメントが既に文字列"さようなら"を含んでいると、ドキュメントは文字列"さようなら　また、会いましょう"を含み、その後にキャリッジリターンが続きます。

```
C_TIME(vhDocRef)
vhDocRef:=Append document("Note.txt") `Noteドキュメントを開く。
SEND PACKET(vhDocRef;" また会いましょう\r") ` 文字列を追加する。
CLOSE DOCUMENT(vhDocRef) `ドキュメントを閉じる。
```

CLOSE DOCUMENT

CLOSE DOCUMENT (DocRef)

引数	型		説明
DocRef	DocRef	→	ドキュメント参照番号

説明

CLOSE DOCUMENTはDocRefで指定したドキュメントを閉じます。

ドキュメントを閉じることは、ファイルに書き込んだデータを確実に保存する唯一の方法です。コマンドOpen document、Create documentまたはAppend documentで開いたドキュメントはすべて、必ずこのコマンドを使用して閉じなければなりません。

例題

以下の例を使用して、新しいドキュメントを作成し、文字列"こんにちは" を書き込み、ドキュメントを閉じます。

```
C_TIME(vhDocRef)
vhDocRef:=Create document("")
If(OK=1)
   SEND PACKET(vhDocRef;"こんにちは")  `ドキュメントに1つの言葉を書き込む
   CLOSE DOCUMENT(vhDocRef)  `ドキュメントを閉じる
End if
```

Convert path POSIX to system

Convert path POSIX to system (posixPath {; *}) -> 戻り値

引数	型		説明
posixPath	テキスト	→	POSIXパス名
*	演算子	→	エンコーディングオプション
戻り値	テキスト	↩	システムシンタックスで表現されたパス名

説明

Convert path POSIX to system コマンドはPOSIX (Unix) シンタックスで表現されたパス名をシステムシンタックスで表現されたパス名に変換します。

posixPath引数にPOSIXシンタックスで表現された、ファイルやフォルダの完全パス名を渡します。これは ("/"から始まる) 絶対パスでなければなりません。ディスクパスを渡さなければならず、(例えばftp://ftp.mysite.frなどから始まる) ネットワークパスを渡すことはできません。

コマンドは現在のシステムシンタックスで表現された、ファイルやフォルダの完全パス名を返します。

オプションの * 引数を使用してposixPath引数がエンコードされているかどうかを指定できます。posixPath引数がエンコードされている場合、この引数を渡さなければなりません。そうでなければ正しく変換されません。このコマンドはエンコードなしのパス名を返します。

例題 1

Mac OSでの例題:

```
$path:=Convert path POSIX to system("/Volumes/machd/file 2.txt")
 // "machd:file 2.txt"を返す
$path:=Convert path POSIX to system("/Volumes/machd/file%202.txt";*)
 // "machd:file 2.txt"を返す
$path:=Convert path POSIX to system("/file 2.txt")
 // "machd:file 2.txt"を返す (machdは起動ディスク)
```

例題 2

Windowsでの例題:

```
$path:=Convert path POSIX to system("c:/docs/file 2.txt")
 // "c:\docs\truc 2.txt" を返す
$path:=Convert path POSIX to system("c:/docs/file%202.txt";*)
 // "c:\docs\truc 2.txt" を返す
```

Convert path system to POSIX

Convert path system to POSIX (systemPath {; *}) -> 戻り値

引数	型		説明
systemPath	テキスト	⇒	システムシンタックスで表現された、相対または絶対パス名
*	演算子	⇒	エンコーディングオプション
戻り値	テキスト	↩	POSIXシンタックスで表現された絶対パス名

説明

Convert path system to POSIX コマンドはシステムシンタックスで表現されたパス名をPOSIX (Unix) シンタックスで表現されたパス名に変換します。

systemPath引数には、システムシンタックスで表現した、ファイルやフォルダのパス名を渡します (Mac OS やWindows)。このパスは絶対パス、あるいはデータベースフォルダ (データベースストラクチャファイルを含むフォルダ) からの相対パスです。コマンド実行時に、パスの要素が実際にディスク上に存在する必要はありません。コマンドはパス名の妥当性をテストしません。

コマンドはファイルやフォルダのPOSIXシンタックスで表現した完全パス名を返します。systemPathに渡されたパスのタイプにかかわらず、コマンドは常に絶対パス名を返します。systemPathに相対パス名を渡すと、4Dはデータベースフォルダのパス名を追加して返します。

オプションの * 引数を使用してPOSIXのエンコーディングを指定できます。デフォルトで、Convert path system to POSIXはPOSIXパスの特別文字をエンコードしません。*引数を渡すと、特別文字は変換されます (例えば"My folder"は"My%20folder"になります)。

例題 1

Mac OSでの例題

```
 $path:=Convert path system to POSIX("machd:file 2.txt")
  // "/Volumes/machd/file 2.txt" を返す (machdが起動ディスクでも)
 $path:=Convert path system to POSIX("machd:file 2.txt";*)
  // "/Volumes/machd/file%202.txt" を返す
 $path:=Convert path system to POSIX("resources:images")
  // "/Volumes/machd/bases/basevideo/resources/images" を返す
```

例題 2

Windowsでの例題

```
 $path:=Convert path system to POSIX("c:\docs\file 2.txt")
  // "c:/docs/file 2.txt" を返す
 $path:=Convert path system to POSIX("\\srv\tempo\file.txt")
  // "//srv/tempo/file.txt" を返す
```

COPY DOCUMENT

COPY DOCUMENT (sourceName ; destinationName {; newName} {; *})

引数	型		説明
sourceName	文字	→	コピーするファイルやフォルダーのパス名
destinationName	文字	→	ファイルやフォルダーコピーの配置先名またはパス名
newName	文字	→	コピーされたファイルまたはフォルダの新しい名前
*	演算子	→	存在する場合、既存のドキュメントを上書きする

説明

COPY DOCUMENTコマンドを使用して、sourceNameによって指定されたファイルやフォルダーをdestinationNameによって指定された場所にコピーします。

- **ファイルのコピー**
  この場合sourceName引数には以下のどちらかを渡す事ができます。
  - ボリュームのルートから始まる完全なファイルパス名
  - データベースフォルダーから相対的なパス名

  destinationName引数には複数のタイプで場所を指定できます:
  - ボリュームのルートから始まる完全なファイルパス名。ファイルはこの場所にコピーされます。
  - ファイル名または相対的なファイルパス名。ファイルはデータベースフォルダー内にコピーされます。サブフォルダーはコマンド実行前に存在していなければなりません。
  - データベースフォルダーから相対的な完全フォルダーパス名またはパス名(destinationNameはプラットフォームに応じたフォルダー区切り文字で終わっていなければなりません)。ファイルは指定したフォルダーにコピーされます。これらのフォルダーはコマンド実行前に存在していなければなりません(自動的に作成されることはありません)。

  オプションの * 引数を渡していない場合、destinationNameで指定した場所にすでにドキュメントが存在するとエラーが生成されます。この引数を渡した場合、既存のファイルは削除され、コピーするドキュメントで置き換えられます。

- **フォルダーのコピー**
  フォルダーを指定するためには、sourceName と destinationName に渡す文字列がプラットフォームに対応したフォルダー区切り文字で終わっていなければなりません。例えばWindows環境下では "C:\\Element\\"はフォルダーを表し、"C:\\Element"はファイルを表します。
  フォルダーをコピーするには、その完全パス名をsourceNameに渡します。このフォルダーは既に存在しなければなりません。フォルダーをsourceName引数に設定した場合、destinationName引数もフォルダーでなければなりません。このとき、ディスクにすでに存在する完全フォルダーパス名を渡さなければなりません。
  sourceName で指定したフォルダーと同じ名前のフォルダーがdestinationName で指定した場所に既に存在し、そのフォルダーが空でない場合、4Dは項目をコピーする前にフォルダーの内容を確認します。オプションの * 引数を渡していない場合、同じ名前のファイルが既に存在すればエラーが生成されます。この引数を渡した場合、その場所のファイルは削除され、コピーするドキュメントで置き換えられます。

ファイルをフォルダーにコピーするために、sourceName にファイルを、destinationName にフォルダーを渡すことは可能です。

任意の newName 引数を渡した場合、コピーした場所のドキュメント(ファイルまたはフォルダー)を改名します。ファイルのコピーの際に渡した場合は、この引数は destinationName 引数に渡された名前を置き換えます。

例題 1

次の例を使用して、そのドキュメントがあるフォルダ内でドキュメントを複製します。

```
COPY DOCUMENT("C:\\FOLDER\\DocName";"C:\\FOLDER\\DocName2")
```

例題 2

次の例を使用して、ドキュメントをデータベースフォルダにコピーします(表示されているC:\\FOLDERは、データベースフォルダではありません) 。

```
COPY DOCUMENT("C:\\FOLDER\\DocName";"DocName")
```

例題 3

次の例を使用して、ドキュメントを一つのボリュームから他のボリュームへコピーします。

```
COPY DOCUMENT("C:\\FOLDER\\DocName";"F:\\Archives\\DocName.OLD")
```

例題 4

次の例を使用して、そのドキュメントがあるフォルダ内で既存のコピーを上書きして、ドキュメントを複製します。

```
COPY DOCUMENT("C:\\FOLDER\\DocName";"C:\\FOLDER\\DocName2";*)
```

例題 5

指定したフォルダーに同じ名前でファイルをコピー:

```
COPY DOCUMENT("C:\\Projects\\DocName";"C:\\Projects\\")
```

例題 6

指定したフォルダーに同じ名前でファイルをコピー。既存のファイルは上書きする:

```
COPY DOCUMENT("C:\\Projects\\DocName";"C:\\Projects\\"; *)
```

例題 7

フォルダーを他のフォルダーにコピー (両フォルダーは既に存在しなければなりません):

```
COPY DOCUMENT("C:\\Projects\\";"C\\Archives\\2011\\")
```

例題 8

以下の例はデータベースフォルダー内に異なるファイルやフォルダーを作成します(ただしこれはWindows用の例題です)。どの場合においても、"folder2"というフォルダーが先に存在していなければなりません。

```
COPY DOCUMENT("folder1\\name1";"folder2\\")
 //"folder2/name1" という名前のファイルを作成します。

COPY DOCUMENT("folder1\\name1";"folder2\\" ; "new")
 //"folder2/new" という名前のファイルを作成します。

COPY DOCUMENT("folder1\\name1";"folder2\\name2")
 //"folder2/name2" という名前のファイルを作成します。

COPY DOCUMENT("folder1\\name1";"folder2\\name2";"new")
 //"folder2/new" という名前のファイルを作成します(name2は無視されます)。

COPY DOCUMENT("folder1\\" ; "folder2\\")
 //"folder2/folder1/" という名前のフォルダを作成します。

COPY DOCUMENT("folder1\\" ; "folder2\\" ; "new")
 //"folder2/new/" という名前のファイルを作成します。
```

CREATE ALIAS

CREATE ALIAS (targetPath ; aliasPath)

引数	型	説明
targetPath	文字 ⇒	エイリアス/ショートカットターゲットのアクセスパスまたは名前
aliasPath	文字 ⇒	エイリアスまたはショートカットの完全なパス名または名前

説明

CREATE ALIASコマンドを使用して、targetPathに渡した目的のファイルまたはフォルダのエイリアス(Windowsでは"ショートカット" と呼ばれる) を作成します。名前と場所は、引数targetPathによって決定されます。

エイリアスは、あらゆる種類のドキュメントやフォルダに役立ちます。エイリアスのアイコンはターゲットアイテムと同じです。アイコンの左下には小さな矢印があります。Mac OSではアイコン名はイタリック文字で表示されます。

このコマンドは、名前をデフォルトで割り当てませんので、引数aliasPathに名前を渡す必要があります。名前だけをこの引数に渡すと、エイリアスがカレントのワーキングフォルダ内に作成されます(通常、ストラクチャファイルを格納しているファイル) 。

Note: Windowsでは、ショートカットは、拡張子".LNK" (可視できません) によって指定されます。この拡張子を渡さなければ、コマンドが自動的に拡張子を付加します。

targetPathに空の文字列を渡すと、コマンドは何もしません。

例題

データベースは、データベースのフォルダ内でソートされた"レポート番号" と呼ばれるテキストファイルを作成します。これらのレポートのショートカットを作成し、便利な場所に保存します。

```
 `メソッドCREATE_REPORT
C_TEXT($vtRport)
C_STRING(250;$vtpath)
C_STRING(80;$vaname)
C_TIME(vDoc)
C_INTEGER($ReportNber)

$vTReport:=...  `レポートを作成する
$ReportNber:=...  `レポート番号を計算する
$vaName:="Report"+String($ReportNber)+".txt"  `ファイル名
vDoc:=Create document($vaName)
If(OK=1)
   SEND PACKET(vDoc;$vTReport)
   CLOSE DOCUMENT(vDoc)
 `エイリアスを追加する
   CONFIRM("Create an alias for this report?")
   If(OK=1)
      $vtPath:=Select folder("Where do you want the alias to be created?")
      If(OK=1)
         CREATE ALIAS($vaName;$vtPath+$vaName)
         If(OK=1)
            SHOW ON DISK($vtPath+$vaName)
 `エイリアスの場所を表示する
         End if
      End if
   End if
End if
```

システム変数およびセット

コマンドの実行が成功すると、OKシステム変数に1が代入されます。その他の場合は、0が代入されます。

Create document

Create document (document {; fileType}) -> 戻り値

引数	型		説明
document	文字	→	ドキュメント名、または 完全なドキュメントのパス名、または 標準ファイルダイアログボックスの空の文字列
fileType	文字	→	スクリーンされるドキュメントタイプのリスト、または ドキュメントをスクリーンしない場合 "*"
戻り値	DocRef	↩	ドキュメント参照番号

説明

Create documentコマンドは新しいドキュメントを作成し、ドキュメント参照番号を返します。

documentには新しいドキュメントの名前、または完全なパス名を渡します。documentが既にディスクに存在する場合、それは上書きされます。しかしdocument がロックされていたり既に開いている場合は、エラーが生成されます。

documentに空の文字列を渡すと、別名で保存ダイアログボックスが表示され、作成したいドキュメントの名前を入力できます。ダイアログをキャンセルした場合ドキュメントは作成されません。**Create document**はヌルDocRefを返し、OK変数に0を代入します。

ドキュメントが正しく作成されそのドキュメントが開かれると、**Create document**はそのドキュメント参照番号を返し、OK変数に1を代入します。システム変数Documentが更新され、作成したドキュメントの完全なアクセスパスを返します。

別名で保存ダイアログボックスを使用してもしなくても、**Create document**はTXT (Windows) または TEXT (Macintosh) ドキュメントをデフォルトで作成します。他のタイプのドキュメントを作成したい場合、引数fileTypeを渡します。

引数fileTypeには、開いているダイアログボックスで選択できるファイルのタイプを渡します。; (セミコロン) で区別される幾つかのタイプのリストを渡すこともできます。それぞれのタイプセットに対して、ダイアログボックスでタイプを選ぶために使用されるメニューにアイテムが追加されます。

Mac OSでは、標準的なMac OSタイプ(TEXT、APPLなど)、またはUTI (Uniformタイプ識別子) タイプのいずれかを渡します。ファイルタイプの標準化のニーズを満たすために、UTIはAppleによって定義されます。例えば、"public.text" は、テキストタイプファイルのUTIタイプになります。UTIについての詳細は、以下のアドレスを参照してください。
https://developer.apple.com/library/mac/#documentation/Miscellaneous/Reference/UTIRef/Articles/System-DeclaredUniformTypeIdentifiers.html

Windowsでは、標準的なMac OSファイルタイプやファイル拡張子 (.txt、.exeなど) を渡します。Windowsでは、ダイアログボックスに"*.*" を入力して全てのファイルタイプを表示させることができます。しかしこの場合、4Dは選択されたファイルタイプの追加チェックを実行します。認められていないファイルタイプを選択すると、コマンドはエラーを返します。
表示されているファイルを1つ以上のタイプに制限したくない場合、文字列"*" (アスタリスク) または".*" を fileTypeに渡します。

Windowsでは、Windowsのファイル拡張子を渡すか、MAP FILE TYPESメカニズムを通してマップされたMacintoshのファイルタイプを渡します。拡張子のないドキュメントや、複数の拡張子を含むドキュメント、そして3文字以上から成る拡張子を含むドキュメントを作成したい場合、引数type を使用せずに、完全な名前をdocument渡します(例2を参照) 。

ドキュメントを作成してドキュメントを開くと、RECEIVE PACKETとSEND PACKETコマンドを使用してドキュメントを読んだり書いたりできます。これらのコマンドにGet document positionとSET DOCUMENT POSITIONコマンドを組み合わせることができます。これにより、ドキュメントのあらゆる箇所に直接アクセスすることが可能となります。

ドキュメントに対して、最後にCLOSE DOCUMENTを呼び出すことを忘れないでください。

例題 1

次の例を使用して、Noteと呼ばれる新しいドキュメントを作成して開きます。文字列"Hello" をそこへ書き込み、ドキュメントを閉じます。

```
C_TIME(vhDoc)
vhDoc:=Create document("Note.txt") `Noteと呼ばれる新しいドキュメントを作成します。
If(OK=1)
   SEND PACKET(vhDoc;"Hello") `ドキュメントで1つの単語を書き込みます。
   CLOSE DOCUMENT(vhDoc) `ドキュメントを閉じます。
End if
```

例題 2

次の例を使用して、Windowsで非標準拡張子を付けてドキュメントを作成します。

```
$vtMyDoc:=Create document("Doc.ext1.ext2") `複数の拡張子
$vtMyDoc:=Create document("Doc.shtml") `長い拡張子
$vtMyDoc:=Create document("Doc.") `拡張子なし (ピリオド "." は必須)
```

システム変数およびセット

ドキュメントが正しく作成されると、システム変数OKに1が代入されます。システム変数Documentは、完全なパス名とdocumentの名前を格納します。

CREATE FOLDER

CREATE FOLDER (folderPath {; *})

引数	型		説明
folderPath	文字	→	作成する新しいフォルダーのパス名
*	演算子	→	パス中に存在しないフォルダーを作成する

説明

CREATE FOLDERコマンドを使用して、folderPathに渡すパス名に応じてフォルダーを作成します。

folderPathに名前を渡すと、データベースフォルダー内にフォルダーが作成されます。

folderPathには他にもボリュームのルートやデータベースフォルダーから開始したフォルダー階層を渡すことができます。この場合引数はフォルダー区切り文字で区切られていなければなりません。
* 引数を省略した場合、階層と中のフォルダーが存在しなければエラーが生成され、フォルダーは作成されません。
CREATE FOLDERコマンドに * 引数を渡すと、必要に応じてフォルダー階層が作成されます。この場合、folderPathにドキュメントパス名を渡すこともできます。ドキュメント名は無視され、folderPathに指定されたフォルダー階層が再帰的に作成されます。

例題 1

次の例を使用して、"Archives" フォルダーをデータベースのフォルダー内に作成します。

```
CREATE FOLDER("Archives")
```

例題 2

次の例を使用して、"Archives" フォルダーをデータベースフォルダーに作成し、"January" および "February" というサブフォルダーを作成します。

```
CREATE FOLDER("Archives")
CREATE FOLDER("Archives\\January")
CREATE FOLDER("Archives\\February")
```

例題 3

次の例を使用して、"Archives" フォルダーをCボリュームのルートレベルに作成します。

```
CREATE FOLDER("C:\\Archives")
```

例題 4

"C:\Archives\2011\January\" フォルダー階層を作成:

```
CREATE FOLDER("C:\\Archives\\2011\\January\\";*)
```

例題 5

既存の"C:\Archives\"フォルダーに"\February\"サブフォルダーを作成:

```
CREATE FOLDER("C:\\Archives\\2011\\February\\Doc.txt";*)
 // "Doc.txt"は無視される
```

DELETE DOCUMENT

DELETE DOCUMENT (document)

引数	型	説明
document	文字	→ ドキュメント名、または 完全なドキュメントのパス名

説明

DELETE DOCUMENTコマンドを使用して、documentに渡したドキュメント名を持つドキュメントを削除します。

ドキュメント名や入力されたパス名が正しくないと、エラーが生成されます。開かれたドキュメントの削除を試みた場合もエラーが発生します。

DELETE DOCUMENT はdocumentに対して空の文字列の引数を受け入れません。空の文字列が使用されると、ファイルを開くダイアログボックスは表示されず、エラーが生成されます。

警告: DELETE DOCUMENT を使用して、ディスク上のあらゆるファイルを削除することができます。他のアプリケーションで作成されたドキュメント、およびアプリケーション自体も含まれます。DELETE DOCUMENT を使用する際は細心の注意を払ってください。ドキュメントの削除は恒久的に作用し、取り消すことはできません。

例題 1

次の例を使用して、Noteという名前のドキュメントを削除します。

```
DELETE DOCUMENT("Note") ` ドキュメントを削除する。
```

例題 2

コマンド APPEND DATA TO PASTEBOARDの例を参照。

システム変数およびセット

ドキュメントの削除は、OKシステム変数に1を代入します。DELETE DOCUMENT がドキュメントを削除できない場合、OKシステム変数に0が代入されます。

DELETE FOLDER

DELETE FOLDER (folder)

引数	型		説明
folder	文字	→	削除されるフォルダの完全なパス名

説明

DELETE FOLDERコマンドを使用して、folder に渡した完全なパスまたは名前を持つフォルダを削除します。

空のフォルダのみが、このコマンドによって削除されます。

- ファイルを格納しているフォルダを削除しようとすると、エラーコード -47が生成されます(空でないフォルダの削除の試み) 。
- folderにファイルのパス、空の文字列、存在しないフォルダのパスを渡すと、コマンドは何もせずエラーコード -43 が生成されます(ファイルが見つかりません) 。

ON ERR CALLコマンドでインストールされたメソッドを通して、これらのエラーを検知することができます。

Document creator

Document creator (document) -> 戻り値

引数	型		説明
document	文字	→	ドキュメントの名前、または ドキュメントの完全なパス名
戻り値	文字	←	空の文字列 (Windows)、または ファイルクリエータ (Mac OS)

説明

Document creatorコマンドは、documentに渡した名前またはパス名を持つドキュメントのクリエータを返します。

Windowsでは、Document creator は空の文字列を返します。

DOCUMENT LIST

DOCUMENT LIST (pathname ; documents {; options})

引数	型		説明
pathname	文字	→	ボリューム、ディレクトリ、またはフォルダーへのパス名
documents	テキスト配列	←	この場所にあるドキュメントの名前
options	倍長整数	→	取得するリストを指定するオプション

説明

DOCUMENT LISTコマンド は、pathnameに渡すパス名にあるドキュメントの名前を要素とするテキスト配列documentsを生成します。

注: 引数pathname は絶対パス名だけを受け入れます。

options 引数を省略した場合、デフォルトで documents 配列にドキュメント名のみが返されます。options 引数にSystem Documentsテーマの以下の定数を渡すと、この動作を変更できます:

定数	型	値	コメント
Absolute path	倍長整数	2	documents配列は絶対パス名を格納
Ignore invisible	倍長整数	8	不可視ドキュメントをリストに含めない
Posix path	倍長整数	4	documents配列はPOSIXフォーマットパス名を格納
Recursive parsing	倍長整数	1	documents配列は指定したフォルダーに含まれるすべてのファイルとサブフォルダーを格納

注:

- 相対モードにおけるRecursive parsingオプション (option 1のみ) では、プラットフォームに応じてサブフォルダー内のドキュメント名は":"または"\"から始まります。
- 相対モードにおけるPosix pathオプション (option 4のみ) では、パスは"/"で始まりません。
- 絶対モードにおけるPosix pathオプション (option 4 + 2) では、パスは常に"/"で始まります。

指定した場所にドキュメントがない場合、コマンドは空の配列を返します。pathnameに渡したパス名が無効だと、**DOCUMENT LIST**はファイルマネジャエラーを生成します。このエラーはON ERR CALLでインストールされるエラー処理メソッドを使用して、検知することができます。

例題 1

フォルダー中のすべてのドキュメントをリスト (デフォルトシンタックス):

```
DOCUMENT LIST("C:\\";arrFiles)
```

```
-> arrFiles:
    Text1.txt
    Text2.txt
```

例題 2

絶対モードでフォルダー中のすべてのドキュメントをリスト:

```
DOCUMENT LIST("C:\\";arrFiles; Absolute path)
```

```
-> arrFiles:
    C:\Text1.txt
    C:\Text2.txt
```

例題 3

再帰 (相対) モードですべてのドキュメントをリスト:

```
DOCUMENT LIST("C:\\";arrFiles;Recursive parsing)
```

```
-> arrFiles:
    Text1.txt
    Text2.txt
    \Folder1\Text3.txt
    \Folder1\Text4.txt
    \Folder2\Text5.txt
    \Folder2\Folder3\Picture1.png
```

例題 4

再帰 (絶対) モードですべてのドキュメントをリスト:

```
DOCUMENT LIST("C:\\MyFolder\\";arrFiles;Recursive parsing+Absolute path)
```

```
-> arrFiles:
    C:\MyFolder\MyText1.txt
    C:\MyFolder\MyText2.txt
    C:\MyFolder\Folder1\MyText3.txt
    C:\MyFolder\Folder1\MyText4.txt
    C:\MyFolder\Folder2\MyText5.txt
    C:\MyFolder\Folder2\Folder3\MyPicture1.png
```

例題 5

再帰POSIX (相対) モードですべてのドキュメントをリスト:

```
DOCUMENT LIST("C:\\MyFolder\\";arrFiles;Recursive parsing+Posix path)
```

```
-> arrFiles:
    MyText1.txt
    MyText2.txt
    Folder1/MyText3.txt
    Folder1/MyText4.txt
    Folder2/MyText5.txt
    Folder2/Folder3/MyPicture1.png
```

Document to text

Document to text (fileName {; charSet {; breakMode}}) -> 戻り値

引数	型		説明
fileName	文字	→	ドキュメント名またはドキュメントへのパス名
charSet	テキスト, 倍長整数	→	文字コード名の名前または数字
breakMode	倍長整数	→	改行の処理モード
戻り値	テキスト	←	ドキュメントから取得したテキスト

説明

Document to textコマンドは、ディスク上のファイルの中身を、4Dのテキスト変数またはテキストフィールドへと直接取り出すことができます。

fileName 引数には、読みだしたいファイル名またはパス名を渡します。ファイルはディスク上に存在している必要があり、そうでない場合にはエラーが生成されます。渡せるものは以下の通りです:

- ファイル名のみ。例えば "myFile.txt" など。この場合、ファイルはアプリケーションのストラクチャーファイルの隣にある必要があります。
- アプリケーションのストラクチャファイルからの相対パス。例えばWindowsでは "\\docs\\myFile.txt" またはOS Xでは ":docs:myFile.txt"
- 絶対パス。例えばWindowsでは "c:\\app\\docs\\myFile.txt" またはOS Xでは "MacHD:docs:myFile.txt"

charSet 引数には、ファイルの中身を読みだす際の文字コードを渡します。標準の文字コード名(例えば"ISO-8859-1" や "UTF-8")を渡す事もできますし、文字コードの MIBEnum ID (倍長整数)を渡す事もできます。4Dによってサポートされている文字コードの一覧の詳細な情報に関しては、CONVERT FROM TEXT コマンドの詳細を参照して下さい。

ドキュメントがバイトオーダーマーク(BOM)を含んでいる場合、4DはcharSet 引数で指定された文字コードのかわりにバイトオーダーマークが指定した文字コードを使用します(つまり、この引数は無視されます)。

ドキュメントがBOMを含まず、charSet set引数が省略されていた場合、4Dはデフォルトで以下の文字コードを使用します。

- Windows: ANSI
- OS X: MacRoman

breakMode 引数には、ドキュメントの改行文字の処理を指示する倍長整数を渡します。"System Documents"テーマ内にある、以下の定数のどれかを渡すことが出きます。

定数	型	値	コメント
Document unchanged	倍長整数	0	何も処理をしません。
Document with CR	倍長整数	3	改行は Mac OSフォーマット(CR、キャリッジリターン)へと変換されます。
Document with CRLF	倍長整数	2	改行はWindowsフォーマット(CRLF、キャリッジリターン＋ラインフィード)へと変換されます。
Document with LF	倍長整数	4	改行はUnixフォーマット(LF、ラインフィード)へと変換されます。
Document with native format	倍長整数	1	改行はOSのネイティブフォーマットに変換されます。Mac OS環境ではCR(キャリッジリターン)に、Windows環境ではCRLF(キャリッジリターン＋ラインフィード)に変換されます。(デフォルトの動作)

デフォルトでは、breakMode 引数を省略した場合、改行はネイティブモード(1)にて処理されます。

Note: このコマンドはOK変数を変更しません。失敗した場合には ON ERR CALL コマンドによって実装されたメソッドを使用することによって割り込み可能なエラーが生成されます。

例題

以下のテキストドキュメントが与えられている場合を考えます(フィールドはtabによって分けられています)。

```
id    name    price    vat
3    4D Tags    99    19.6
```

以下のコードを実行すると、

```
 $Text:=Document to text("products.txt")
```

... 次のような結果が得られます。

```
  // $Text = "id\tname\tprice\tvat\r\n3\t4D Tags\t99 \t19.6"
  // \t = tab
  // \r = CR
```

FOLDER LIST

FOLDER LIST (pathname ; directories)

引数	型		説明
pathname	文字	→	ボリュームのパス名、ディレクトリ、またはフォルダ
directories	文字配列	←	ロケーションにあるディレクトリの名前

説明

FOLDER LIST コマンドは、pathnameに渡すパス名にあるフォルダの名前を要素とするテキストまたは文字列配列directoriesを生成します。

Note: 引数pathname は完全なパス名だけを受け入れます。

指定した場所にフォルダがない場合、コマンドは空の配列を返します。pathnameに渡すパス名が無効だと、FOLDER LISTはファイルマネジャエラーを生成します。このエラーはON ERR CALLメソッドを使用して検知することができます。

GET DOCUMENT ICON

GET DOCUMENT ICON (docPath ; icon {; size})

引数	型		説明
docPath	文字	→	アイコンを取得するドキュメントのパスまたは名前、または空の文字列の場合標準のファイルを開くダイアログ
icon	ピクチャー	→	ピクチャ変数またはフィールド
size	倍長整数	→	返されたピクチャのサイズ(ピクセルで)

説明

GET DOCUMENT ICONコマンドは、filePathに渡した名前または完全なパス名を持つドキュメントのアイコンを4Dのピクチャ変数またはフィールドiconに返します。filePath はあらゆるタイプのファイル (実行ファイル、ドキュメント、ショートカットまたは別名など) またはフォルダを指定します。

filePathには、完全なドキュメントのパス名が含まれていなければなりません。あるいはドキュメント名または相対パス名を渡すこともできます。しかしこの場合、ドキュメントはカレントのワーキングディレクトリに存在していなければなりません (通常、データベースストラクチャファイルを含んでいるフォルダ) 。

空の文字列をfilePathに渡すと、標準的なファイルを開くダイアログボックスが表示され、ユーザは読み込むファイルを選択することができます。ダイアログボックスが受け入れられると、Documentシステム変数は選択されたファイルの完全なパス名を格納します。

4Dのピクチャフィールドまたは変数をiconに渡します。コマンドが実行されるとこの引数にはファイル (PICTフォーマット) のアイコンが返されます。

任意の引数sizeによって、返されたアイコンのサイズをピクセルで設定できます。この値は、実際にアイコンを含む正方形の両方の側面の長さを表します。通常アイコンは32x32ピクセル(大きいアイコン) または16x16ピクセル(小さいアイコン) で定義されます。0を渡したり、この引数を省略する場合、使用できる最大のアイコンが返されます。

Get document position

Get document position (DocRef) -> 戻り値

引数	型		説明
DocRef	DocRef	→	ドキュメント参照番号
戻り値	実数	↩	ドキュメント開始位置からの ファイル位置(バイト単位)

説明

このコマンドはDocRefに渡したドキュメント参照番号を持つ、現在開いているドキュメントだけに機能します。

Get document positionは、ドキュメントの最初から見て、次の読み込み(RECEIVE PACKET) または書き込み(SEND PACKET) が発生する位置を返します。

GET DOCUMENT PROPERTIES

GET DOCUMENT PROPERTIES (document ; locked ; invisible ; created on ; created at ; modified on ; modified at)

引数	型		説明
document	文字	→	ドキュメントの名前
locked	ブール	←	Trueの場合はロック、またはFalseの場合はアンロック
invisible	ブール	←	Trueの場合は非表示、またはFalseは表示
created on	日付	←	作成日
created at	時間	←	作成時間
modified on	日付	←	更新日
modified at	時間	←	更新時間

説明

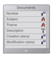
GET DOCUMENT PROPERTIESコマンドは、引数documentに渡した名前またはパス名を持つドキュメントに関する情報を返します。

呼び出し後、以下の情報が返されます。

- ドキュメントがロックされていると、lockedはTrueを返します。ロックされているドキュメントを修正することはできません。
- ドキュメントが非表示の場合、invisibleはTrueを返します。
- created onとcreated atにはドキュメントが作成された日付と時間が返されます。
- modified onとmodified atにはドキュメントが最後に修正された日付と時間が返されます。

例題

ドキュメントデータベースを作成し、そのデータベースに作成したすべてのレコードをディスク上のドキュメントに書き出す場合を想定します。データベースは定期的に更新されているため、ドキュメントが存在しなかったり、ドキュメントが最後に保存された後に、対応するレコードが修正されてしまった場合には、各ドキュメントを作成、再作成するようなデータ書き出しのアルゴリズムを記述します。そのため、ドキュメント(存在する場合には) を修正した日付と時間をそれに対応するレコードと比較する必要があります。これを図解するために、以下のテーブルを使用してその定義を表します。



各レコードに日付と時間の両方を保存するのではなく、任意の日付時刻とレコードが保存された日付時間との間の経過数秒を示す"タイムスタンプ"の値を保存することができます (この例では1995年1月1日午前0時を使用しています) 。

この例では、[Documents]Creation Stampフィールドにレコードが最初に作成されたときのタイムスタンプがあり、[Documents]Modification Stampフィールドにレコードが最後に更新されたときのタイムスタンプがあります。

この後に示されているTime stampプロジェクトメソッドでは、引数が渡されない場合には、現在の日付と時間または特定の日付と時間としてタイムスタンプを計算します。

```
  `Time stamp プロジェクトメソッド
  `Time stamp { ( 日付 ; 時間 ) } -> 倍長整数
  `Time stamp { ( 日付 ; 時間 ) } -> 1995年1月1日から経過した数秒

 C_DATE($1;$vdDate)
 C_TIME($2;$vhTime)
 C_LONGINT($0)

 If(Count parameters=0)
    $vdDate:=Current date
    $vhTime:=Current time
 Else
    $vdDate:=$1
    $vhTime:=$2
 End if
 $0:=(($vdDate-!01/01/95!)*86400)+$vhTime
```

注: このメソッドを使用すると、日付と時間を1995/01/01 00:00:00から2063/01/19 03:14:07の間でコード化できるため、倍長整数の0から2^31 -1の範囲に対応できます。

一方、次に示されているTime stamp to dateおよびTime stamp to timeのプロジェクトメソッドでは、タイムスタンプに保存されている日付および時間を抽出することができます。

```
  `Time stamp to dateプロジェクトメソッド
  `Time stamp to date ( 倍長整数 ) -> 日付
  `Time stamp to date ( Time stamp ) -> 抽出する日付

 C_DATE($0)
 C_LONGINT($1)

 $0:=!01/01/95!+($1\86400)


  `Time stamp to timeプロジェクトメソッド
  `Time stamp to time ( 倍長整数 ) -> 日付
  `Time stamp to time ( Time stamp ) -> 抽出する時間

 C_TIME($0)
 C_LONGINT($1)

 $0:=Time(Time string(?00:00:00?+($1%86400)))
```

レコードの作成や更新の方法に関係なく、レコードのタイムスタンプが、正しく更新されるようにするには、[Documents]テーブルのトリガーを使用して、その規則を強制します。

```
  `[Documents]テーブルのトリガー

 Case of
    :(Trigger event=Save New Record Event)
       [Documents]Creation Stamp:=Time stamp
       [Documents]Modification Stamp:=Time stamp
    :(Trigger event=Save Existing Record Event)
       [Documents]Modification Stamp:=Time stamp
 End case
```

このトリガをデータベースに記述すると、以下のCREATE DOCUMENTATIONプロジェクトメソッドの作成に必要となるすべての準備が整います。ドキュメントの作成および更新の日付および時間の処理には**GET DOCUMENT PROPERTIES**コマンドおよびSET DOCUMENT PROPERTIESコマンドを使用します。

```
  `CREATE DOCUMENTATIONプロジェクトメソッド

 C_STRING(255;$vsPath;$vsDocPathName;$vsDocName)
 C_LONGINT($vlDoc)
 C_BOOLEAN($vbOnWindows;$vbDoIt;$vbLocked;$vbInvisible)
 C_TIME($vhDocRef;$vhCreatedAt;$vhModifiedAt)
 C_DATE($vdCreatedOn;$vdModifiedOn)

 If(Application type=4D Client)
  `4D Clientを実行している場合には、4D Clientが存在するクライアントマシンで
  `ドキュメントをローカルに保存します。
    $vsPath:=Long name to path name(Application file)
 Else
  `それ以外の場合には、データファイルが存在する位置にドキュメントを保存する
    $vsPath:=Long name to path name(Data file)
 End if
  `任意で"Documentation"と命名したディレクトリにあるドキュメントを保存する
 $vsPath:=$vsPath+"Documentation"+Char(Directory symbol)
  `このディレクトリが存在しない場合には、作成する
 If(Test path name($vsPath)#Is a folder)
    CREATE FOLDER($vsPath)
 End if
  `古いドキュメント、つまり、対応するレコードが削除されているドキュメントは
  `削除しなければならないため、既にあるドキュメントのリストを作成する
 ARRAY STRING(255;$asDocument;0)
 DOCUMENT LIST($vsPath;$asDocument)
  `[Documents] テーブルから全てのレコードを 選択する
 ALL RECORDS([Documents])
  `各レコードに対して
 $vlNbRecords:=Records in selection([Documents])
 $vlNbDocs:=0
 $vbOnWindows:=On Windows
 For($vlDoc;1;$vlNbRecords)
  ` ディスクにドキュメントを (再) 作成しなければならないと想定する
    $vbDoIt:=True
  `ドキュメントの名前およびパス名を求める
    $vsDocName:="DOC"+String([Documents]Number;"00000")
    $vsDocPathName:=$vsPath+$vsDocName
  `このドキュメントは既に存在するか？
    If(Test path name($vsDocPathName+".HTM")=Is a document)
  `その場合には、ドキュメントのリストからドキュメントを除去します
  `このようにすると、ドキュメントが削除される場合があります
       $vlElem:=Find in array($asDocument;$vsDocName+".HTM")
       If($vlElem>0)
          DELETE FROM ARRAY($asDocument;$vlElem)
       End if
  `レコードが最後に更新された後で、ドキュメントが保存されたか？
       GET DOCUMENT
PROPERTIES($vsDocPathName+".HTM";$vbLocked;$vbInvisible;$vdCreatedOn;$vhCreatedAt;$vdModifiedOn;$vhModifiedAt)
       If(Time stamp($vdModifiedOn;$vhModifiedAt)>=[Documents]Modification Stamp)
  `その場合には、ドキュメントを再作成する必要はない
          $vbDoIt:=False
       End if
    Else
  `ドキュメントは存在しないため、これら2つの変数をリセットし、ドキュメント
  `の最終的なプロパティを設定する前にこれらを計算する必要があることを確認できるようにする
       $vdModifiedOn:=!00/00/00!
       $vhModifiedAt:=?00:00:00?
    End if
  `ドキュメントを (再) 作成する必要があるのか？
    If($vbDoIt)
  `その場合には、更新されたドキュメントの数を増分する
       $vlNbDocs:=$vlNbDocs+1
  `ドキュメントが既に存在する場合には、削除する
       DELETE DOCUMENT($vsDocPathName+".HTM")
  `再度、作成する
       If($vbOnWindows)
          $vhDocRef:=Create document($vsDocPathName;"HTM")
       Else
          $vhDocRef:=Create document($vsDocPathName+".HTM")
       End if
       If(OK=1)
  `ここで、ドキュメントの内容を記述する
          CLOSE DOCUMENT($vhDocRef)
          If($vdModifiedOn=!00/00/00!)
  `ドキュメントは存在しなかったため、
  `更新の日付および時間を正しい値に設定する
             $vdModifiedOn:=Current date
             $vhModifiedAt:=Current time
          End if
  `ドキュメントのプロパティを変更し、作成の日付および時間が対応するレコードと等しくなるようにする
          SET DOCUMENT PROPERTIES($vsDocPathName+".HTM";$vbLocked;$vbInvisible;Time stamp to
date([Documents]Creation Stamp);Time stamp to time([Documents]Creation Stamp);$vdModifiedOn;$vhModifiedAt)
       End if
    End if
  `現在実行中の処理を確認するために
    SET WINDOW TITLE("Processing Document "+String($vlDoc)+" of "+String($vlNbRecords))
    NEXT RECORD([Documents])
 End for
  `古いドキュメント、つまりまた$asDocument配列内にあるドキュメントを削除する
 For($vlDoc;1;Size of array($asDocument))
    DELETE DOCUMENT($vsPath+$asDocument{$vlDoc})
    SET WINDOW TITLE("Deleting obsolete document: "+Char(34)+$asDocument{$vlDoc}+Char(34))
 End for
  ` We're done
 ALERT("Number of documents processed: "+String($vlNbRecords)+"\rNumber of documents updated:
"+String($vlNbDocs)+"\rNumber of documents deleted: "+String(Size of array($asDocument)))
```

Get document size

Get document size (document {; *}) -> 戻り値

引数	型		説明
document	文字, DocRef	→	ドキュメント参照番号 または、ドキュメントの名前
*	演算子	→	Mac OSのみ: 省略した場合、データフォークのサイズ 指定した場合、リソースフォークのサイズ
戻り値	実数	←	ドキュメントのサイズ(バイト単位)

説明

Get document sizeコマンドは、ドキュメントのサイズをバイト単位で表示して返します。

ドキュメントを開いている場合、そのドキュメント参照番号をdocumentに渡します。
ドキュメントを開いていない場合、その名前またはパス名documentに渡します。

Macintoshでは、任意の引数 * を渡さない場合、データフォークのサイズが返されます。引数 * を渡すと、リソースフォークのサイズが返されます。

Get localized document path

Get localized document path (relativePath) -> 戻り値

引数	型		説明
relativePath	テキスト	→	ローカライズされたバージョンを取得したいドキュメントの相対パス名
戻り値	テキスト	←	ローカライズされたドキュメントの絶対パス名

説明

Get localized document path コマンドはxxx.lprojフォルダ内に存在する、相対パスで指定されたドキュメントの完全 (絶対) パス名を返します。

このコマンドは**Resources**フォルダおよびxxx.lproj (xxxは言語コード) サブフォルダが存在するマルチ言語アプリケーションアーキテクチャで使用しなければなりません。このアーキテクチャにおいて、4Dはローカライズされた.xliffタイプや画像などのファイルを自動でサポートします。しかし開発者は他のタイプのファイルについても同じメカニズムを使用する必要があるかもしれません。

relativePathに検索するドキュメントの相対パス名を渡します。渡すパス名はデータベースの"xxx.lproj"フォルダの第一階層から相対でなければなりません。このコマンドはデータベースのカレント言語に対応する "xxx.lproj"フォルダを使用した完全パス名を返します。

Note: カレント言語は**Resources**の内容に基づき4Dが自動で (Get database localizationコマンド参照)、あるいは新しいSET DATABASE LOCALIZATIONコマンドで設定されます。

relativePath引数はシステムまたはPOSIXシンタックスを使用して表現できます。例えば:

- xsl/log.xsl (POSIXシンタックス: Mac OSおよびWindowsで利用可)
- xsl\log.xsl (Windows のみ)
- xsl:log.xsl (Mac OS のみ)

コマンドから返される絶対パス名は常にシステムシンタックスで表されます。

4D Server: リモートモードで、コマンドがクライアントプロセスから呼び出された場合、クライアントマシンの**Resources**フォルダのパスが返されます。

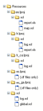
4Dは処理されるマルチ言語アプリケーションの、ありうるすべてのケースのシーケンスを試しながら、ファイルを探します。ステップごとに4Dは言語に対応するフォルダ内でrelativePathの存在を検証し、ファイルを見つければその完全パスを返します。relativePathが見つからないかフォルダが存在しない場合、4D次のステップを試みます。検索ステージ毎のフォルダは以下のようになります:

カレント言語 (例: fr-ca)
- *地域なしのカレント言語 (例: fr)*
  - *開始時にデフォルトでロードされる言語 (例: ja-jp)*
    - *開始時にデフォルトでロードされる地域なし言語 (例: ja)*
      - *最初に見つかった .lproj フォルダ (例: it.lproj)*
        - *Resourcesフォルダの第一レベル*

relativePathがこれらのパスのどこにも存在しない場合、コマンドは空の文字列を返します。

例題

XMLやHTMLファイルを変換する目的で、"log.xsl"変換ファイルを使用したいとします。このファイルはカレント言語により異なるため、どの"log.xsl"ファイルを使用するか決定する必要があります。
Resources フォルダの中身は以下のようになっています:



カレント言語に適用する.xslファイルを決定するには、以下のコードを使用します:

```
$myxsl:=Get localized document path("xsl/log.xsl")
```

カレントの言語が日本語である場合 (ja)、コマンドは以下を返します:

- Windows: C:\users\...\...\...\resources\ja.lproj\xsl\log.xsl
- Mac OS: "HardDisk:users:...:...:...:resources:ja.lproj:xsl:log.xsl"

MAP FILE TYPES

MAP FILE TYPES (macOS ; windows ; context)

引数	型	説明
macOS	文字 ⇒	Mac OSのファイルタイプ(4文字)
windows	文字 ⇒	Windowsのファイル拡張子
context	文字 ⇒	Windowsのファイルを開くダイアログの ファイルの種類ドロップダウンリストに表示される文字列

説明

MAP FILE TYPESを使用して、Windowsのファイル拡張子とMacintoshのファイルタイプを関連づけることができます。

このルーチンを1回呼び出すだけで、データベースによる作業セッション全体についてマッピングを設定することができます。呼び出しが行われると、Windows上で Create document、Append document、そしてOpen resource file コマンドを実行すると、実際に引数としてルーチンに渡されたMacintoshのファイルタイプは、自動的にWindowsの拡張子で置き換えられます。

引数macOS には、4文字のMacintoshのファイルタイプを渡します。4文字の文字列を渡さなければ、コマンドは動作せずエラーが発生します。

引数windowsには、1からX文字のWindowsのファイル拡張子を渡します。1から3文字の文字列を渡さなければ、コマンドは動作せず、エラーが発生します。

引数context には、Windowsのファイルを開くダイアログのファイルの種類ドロップダウンリストに表示される文字列を渡します。コンテキストの文字列は32文字以内に制限されており、それ以上の文字は無視されます。

重要: Windowsのファイル拡張子とMachintoshのファイルタイプとのマッピングを実行すると、その作業セッション中はマッピングを変更、または削除できません。4Dアプリケーションの開発とデバッグの際に変更する必要がある場合は、データベースを開き直して、再度ファイル拡張子のマッピングを行います。

例題

次の4Dのコード(の一部) を使用して、MacintoshのMS-Wordファイルタイプ"WDBN" をWindowsのファイル拡張子".DOC" にマッピングします。

```
MAP FILE TYPES("WDBN";"DOC";"Word documents")
```

上記の呼び出しが実行され次のコードが実行されると、WordドキュメントがWindowsとMacintoshのファイルを開くダイアログに表示されます。

```
$DocRef:=Open document("";"WDBN")
If(OK=1)
  ` ...
End if
```

MOVE DOCUMENT

MOVE DOCUMENT (srcPathname ; dstPathname)

引数	型	説明
srcPathname	文字 ⇒	既存ドキュメントへの完全なパス名
dstPathname	文字 ⇒	移動先のパス名

説明

MOVE DOCUMENTコマンドを使用して、ドキュメントを移動、ドキュメント名を変更します。

srcPathname に既存ドキュメントへの完全なパス名、dstPathnameに新しい名前と位置を指定します。

警告: MOVE DOCUMENTを使用して、同じボリュームのディレクトリの間でドキュメントを移動させることができます。2つの異なるボリュームの間でドキュメントを移動させたい場合、COPY DOCUMENT を使用して、ドキュメントを移動させます。そしてDELETE DOCUMENTを使用してそのドキュメントのオリジナルのコピーを削除します。

例題 1

次の例を使用して、ドキュメントDocNameの名前を変更します。

```
MOVE DOCUMENT("C:\\FOLDER\\DocName";"C:\\FOLDER\\NewDocName")
```

例題 2

次の例を使用して、ドキュメントDocNameを移動させ、名前を変更します。

```
MOVE DOCUMENT("C:\\FOLDER1\\DocName";"C:\\FOLDER2\\NewDocName")
```

例題 3

次の例を使用して、ドキュメントDocNameを移動させます。

```
MOVE DOCUMENT("C:\\FOLDER1\\DocName";"C:\\FOLDER2\\DocName")
```

Note: 最後の2つの例では、移動先のフォルダ"C:\\FOLDER2" が存在していなければなりません。MOVE DOCUMENTコマンド は、ドキュメントを移動させるだけで、フォルダを作成しません。

Open document

Open document (document {; fileType {; mode}}) -> 戻り値

引数	型		説明
document	文字	→	ドキュメント名、または ドキュメントへの完全なパス名、または 空の文字列の場合、標準のファイルダイアログボックス表示
fileType	文字	→	表示されるドキュメントタイプのリスト、または ドキュメントを表示しない"*"
mode	倍長整数	→	ドキュメントを開くモード
戻り値	DocRef	↩	ドキュメント参照番号

説明

Open documentコマンドは、documentに渡した名前またはパス名を持つドキュメントを開きます。

documentに空の文字列を渡した場合、ファイルを開くダイアログボックスが表示され、開くドキュメントを選択できます。このダイアログをキャンセルするとドキュメントは開かれず、**Open document**はドキュメント参照番号に空値を返し、システム変数OKに0を代入します。

- ドキュメントが正しく開かれると、**Open document**はドキュメント参照番号を返し、システム変数OKに1を代入します。
- ドキュメントが既に読み込みのみモードで開かれていて、引数modeが省略されると、**Open document**はデフォルトで読み書きモードでドキュメントを開き、OK変数に1を代入します。
- ドキュメントが読み書きモードで既に開かれていて、書き込みモードで開こうとすると、エラー (-43) が発生します。しかし読み込みのみモードで開くことはでき、この場合OKシステム変数は1に設定されます。
- ドキュメントが存在しない場合、エラーが発生します。

引数fileTypeには、開くダイアログボックスで選択するファイルのタイプを渡します。; (セミコロン) で区切られた幾つかのタイプのリストを渡すことができます。それぞれのタイプセットに対して、ダイアログボックスでタイプを選択する際に使用するメニューに、アイテムが追加されます。
Mac OSでは標準的なMac OSタイプ(TEXT、APPLなど) またはUTI (Uniformタイプ識別子) タイプのどちらかを渡します。ファイルタイプの標準化のニーズを満たすために 、UTIはAppleによって指定されます。例えば、"public.text" は、テキストタイプのファイルのUTIタイプとなります。UTIに関する詳細については、次のアドレスを参照してください。
https://developer.apple.com/library/mac/#documentation/FileManagement/Conceptual/understanding_utis/understand_utis_intro/understand_utis_intro.html
Windowsでは、標準的なMac OSのファイルタイプ (4Dは内部的に対応を行います) またはファイル拡張子(.txt、.exe など) を渡します。Windowsでは、ダイアログボックスに"*.*"を入力することによって、全てのファイルタイプを表示させることができます。しかしこの場合、4Dは選択されたファイルタイプのチェックを追加して実行します。認識されていないファイルタイプが選択された場合、コマンドはエラーを返します。
表示されているファイルを1つ以上のタイプに限定したくない場合は、文字列"*" (アスタリスク) または".*" をfileTypeに渡します。

任意の引数modeは、ドキュメントがどのように開かれるかを指定することができるようにするものです。4つのオープンモードが指定可能です。4D には下記の定数が "System Documents" テーマで定義されています。

定数	型	値
Get Pathname	倍長整数	3
Read and Write	倍長整数	0
Read Mode	倍長整数	2
Write Mode	倍長整数	1

ドキュメントが開かれると、**Open document**はドキュメントの最初にファイルの書き込み/読み込み位置を設定します。一方、Append documentは、ドキュメントの最後にファイルの書き込み/読み込み位置を設定します。

ドキュメントを開くと、RECEIVE PACKETとSEND PACKETコマンドを使用してドキュメントを読んだり、書いたりすることができます。これらのコマンドをGet document positionおよびSET DOCUMENT POSITIONコマンドと組み合わせることにより、ドキュメントのあらゆる箇所へ直接アクセスすることが可能となります。

最後に、開かれたドキュメントに対して、CLOSE DOCUMENTを呼び出すことを忘れないようにしてください。

例題 1

以下の例を使用して、Note.txtという既存のドキュメントを開き、"Good-bye"という文字列を書き込み、そしてドキュメントを閉じます。ドキュメントが既に"Hello" という文字列を含む場合、この文字列は上書きされます。

```
C_TIME(vhDoc)
vhDoc:=Open document("Note.txt";Read and Write) `ノートというドキュメントを開く
If(OK=1)
   SEND PACKET(vhDoc"Good-bye") `ドキュメントに1語書き込む
   CLOSE DOCUMENT(vhDoc) `ドキュメントを閉じる
End if
```

例題 2

ドキュメントがすでに書き込みモード開かれていても、読みをおこなうことはできます。

```
vDoc:=Open document("PassFile";"TEXT") `ファイルが開いている
 `ファイルを閉じる前に、読み込み専用モードで検査できる
vRef:=Open document("PassFile";"TEXT";Read Mode)
```

システム変数およびセット

ドキュメントが正しく開かれると、OKシステム変数に1が代入されます。その他の場合は0が代入されます。呼び出し後、Documentシステム変数には完全なドキュメント名が入っています。
Open documentを3のモードで呼ぶと、関数は?00:00:00? (ドキュメント参照無し) を返します。ドキュメントは開きませんが、DocumentとOKシステム変数は更新されます。

- OKは1と等しい。
- documentには、完全なパス名とそのドキュメントの名前が入っています。

Note: 空の文字列をdocumentに渡した場合はファイルを開くダイアログボックスが表示されます。ドキュメントを選択しOKボタンをクリックすると、documentは選択したドキュメントのパスに設定され、OKに1が代入されます。Cancelボタンをクリックすると、OKに0が代入されます。

RESOLVE ALIAS

RESOLVE ALIAS (aliasPath ; targetPath)

引数	型		説明
aliasPath	文字	→	エイリアス/ショートカットのアクセスパスまたは名前
targetPath	文字	←	エイリアス/ショートカットターゲットのアクセスパスまたは名前

説明

RESOLVE ALIASコマンドは、目的のファイルまたはエイリアス(Windowsではショートカットと呼ばれる) のフォルダへの完全なパスを返します。

エイリアスへの完全なパスをaliasPathに渡します。
このコマンドが実行されると、targetPath変数にはエイリアスのターゲットファイルまたはフォルダへの完全なパスが格納され、OKシステム変数に1が代入されます。
aliasPathに渡されたパスがエイリアスではなくファイルである場合、targetPathはそのファイルのパスを返し、OKシステム変数に0が代入されます。

システム変数およびセット

aliasPath がエイリアスやショートカットの場合、OKシステム変数は1に設定されます。aliasPath が標準のファイルを指している場合、OKシステム変数は0に設定されます。

Select document

Select document (directory ; fileTypes ; title ; options {; selected}) -> 戻り値

引数	型		説明
directory	テキスト, 倍長整数	→	ドキュメント選択ダイアログボックスで ディレクトリのアクセスパスをデフォルトで表示する、または デフォルトユーザフォルダを表示する空の文字列 (Windowsでは"My documents"、Mac OSでは"Documents")、または メモリーされたアクセスパスの番号
fileTypes	テキスト	→	フィルタするドキュメントタイプのリスト、または ドキュメントをフィルタしない"*"
title	テキスト	→	選択ダイアログボックスのタイトル
options	倍長整数	→	任意の選択
selected	テキスト配列	←	アクセスパスのリストを含む配列 + 選択されたファイルの名前
戻り値	文字	↩	選択されたファイルの名前(複数の選択がある場合、リストの最初のファイル)

説明

Select documentコマンドは、標準のドキュメントを開くことのできるダイアログボックスを表示します。このダイアログボックスでは、1つ以上のファイルを選択できます。そしてこのコマンドは選択されたファイルの名前または完全なアクセスパスを返します。

引数directoryは、フォルダの内容がドキュメントを開くダイアログボックスで冒頭に表示されるフォルダを示します。3タイプの値を渡すことができます。

- 表示するフォルダへの完全なアクセスパスを含むテキスト。
- 空の文字列。この場合カレントオペレーティングシステムのデフォルトユーザフォルダ (Windowsでは"My documents"、Mac OSでは"Documents") が設定されます。
- 記憶されたアクセスパスの数値 (1から32,000) 。これは選択ボタンをクリックしたときに開いているフォルダのアクセスパスをメモリに保存できることを意味します。つまり、ユーザによって選択されたフォルダを指します。このコマンドの最初の呼び出しで任意の数値を呼び出すと (例えば5)、コマンドはオペレーティングシステムのデフォルトユーザフォルダを表示します (空の文字列を渡すことに対応する) 。ユーザはハードディスク上の複数のフォルダをざっと見る場合もありますが、選択ボタンをクリックすると、アクセスパスが記憶され、数値の5が結び付けられます。更に数値の5を呼び出すと、記憶されたアクセスパスがデフォルトで使用されます。新しい場所が選択されると、5に対応するパスが更新されます。

このメカニズムを利用して、32,000までのパス名を記憶できます。Windowsではセッションの間だけそれぞれのパスが保持されます。Mac OSでは一つのセッションから次のセッションまで、パスが保持されています。パス名が正しくないと、引数defaultPathは無視されます。

Note: このメカニズムは、Select folderコマンドで使用しているメカニズムと同じです。記憶されたパス名の数値は、この二つのコマンド間で共有されます。

引数fileTypeには、開くダイアログボックスで選択するファイルのタイプを渡し、; (セミコロン) で区切られた幾つかのタイプのリストを渡します。それぞれの指定されたタイプに対して、ダイアログボックスの選択メニューに行が追加されます。

- Mac OSでは、標準的なMac OSタイプ(TEXT、APPLなど) またはUTI (Uniformタイプ識別子) タイプのどちらかを渡します。ファイルタイプの標準化のニーズを満たすために 、UTIはAppleによって指定されます。例えば、"public.text" は、テキストタイプのファイルのUTIタイプとなります。UTIに関する詳細については、次のアドレスを参照してください。
  http://developer.apple.com/mac/library/documentation/FileManagement/Conceptual/understanding_utis/utilist/UTIlist.html
- Windowsでは、標準的なMac OSのファイルタイプ (4Dが内部的に対応させます) またはファイル拡張子 (.txt、.exe など) を渡します。Windowsでは、ダイアログボックスに"*.*"を入力することによって、全てのファイルタイプを表示させることができます。しかしこの場合、4Dは選択されたファイルタイプのチェックを追加して実行します。認識されていないファイルタイプが選択された場合、コマンドはエラーを返します。

表示されているファイルを1つ以上のタイプに限定したくない場合は、文字列"*" (アスタリスク) または"*.*" をfileTypeに渡します。

ダイアログボックスに表示されるラベルを引数titleに渡します。デフォルトで、空の文字列を渡すと、ラベル"開く" が表示されます。

引数optionsはファイルを開くダイアログボックスで使用できる詳細機能を指定します。4D には下記の定数がSystem DocumentsSystem Documentsテーマで定義されています。

定数	型	値	コメント
Alias selection	倍長整数	8	ドキュメントとしてショートカット(Windows) やエイリアス(Mac OS) を選択できます。この定数が使用されずにエイリアスやショートカットが選択されると、コマンドはターゲット要素のアクセスパスを返します。定数を渡すと、コマンドはエイリアスまたはショートカット自体のパスを返します。
File name entry	倍長整数	32	ユーザーに"別名で保存 (新規保存)"ダイアログへのファイル名の入力を許可する。4Dはファイルを作成しません。ファイルの作成は開発者が行わなくてはなりません。Documentシステム変数が更新されます。このコンテキストではdirectory引数にディレクトリではなくファイルへのパスが格納されているかもしれません。そのファイル名は別名で保存テキストフィールドのデフォルトファイル名として使用されます。親ディレクトリはデフォルトパスとして使用されます。
Multiple files	倍長整数	1	キーの組み合わせ**Shift+click** (隣接するファイルの選択) と**Ctrl+click** (Windows) 、または**Command+click** (Mac OS) を使用すると、複数のファイルを同時に選択できます。この場合引数selectedを渡し、選択されたファイルをすべてリストにして受け取ります。この定数が使用されないと、コマンドは複数ファイルの選択を許可しません。
Package open	倍長整数	2	(Mac OSのみ): パッケージをフォルダとして開きその内容を閲覧できます。この定数が使用されないと、コマンドはパッケージの開封を許可しません。
Package selection	倍長整数	4	(Mac OSのみ): パッケージの選択を許可します。この定数が使用されないと、コマンドはソフトウェアパッケージの選択を許可しません。
Use sheet window	倍長整数	16	(Mac OSのみ): 選択ダイアログボックスをシートウィンドウで表示します(Windowsでは、このオプションは無効です)。シートウィンドウはMac OS Xインタフェースに特有なもので、グラフィックアニメーションを伴います (詳細については、ウィンドウタイプを参照してください) 。この定数が使用されないと、コマンドは標準なダイアログボックスを表示します。

オプションを使用したくない場合は、引数optionsに0を渡します。

任意の引数selectedで、ユーザが選択したすべてのファイルの完全なアクセスパス (アクセスパス+名前) を取得します。コマンドはユーザの選択に従って、配列を作成、サイズ決定して、埋めたりします。オプションMultiple filesを使用する場合や選択したファイルのアクセスパスを探したい時に、この引数を使用すると便利です (配列の値からコマンドによって返されたファイルの名前を取り出します) 。ファイルが選択されていない場合、空の配列が返されます。

注: Mac OS上では、選択したパッケージはフォルダとして扱われます。selected 配列に返されるパス名は最後に":"を含みます。例: Disk:Applications:4D:4D v11.4:US:4D Volume Desktop.app:

コマンドは選択されたファイルの名前(Windowsでは名前+拡張子) を返します。複数のファイルが選択されている場合、コマンドは選択されたファイルのリストにある最初のファイル名を返します。ファイルのリストは引数selectedで取得されます。ファイルが何も選択されていない場合、コマンドは空の文字列を返します。

例題 1

次の例を使用して、4Dのデータファイルを特定します。

```
C_LONGINT($platform)
PLATFORM PROPERTIES($platform)
If($platform=Windows)
   $DocType:=".4DD"
Else
   $DocType:="com.4d.4d.data-file" `UTIタイプ
End if
$Options:=Alias selection+Package open+Use sheet window
$Doc:=Select document("";$DocType;"Select the data file";$Options)
```

例題 2

ユーザーが指定したカスタムドキュメントを作成する:

```
$doc:=Select document(System folder(Documents folder)+"Report.pdf";"pdf";"Report name:";File name entry)
If(OK=1)
   BLOB TO DOCUMENT(Document;$blob) // $blob には作成するドキュメントの内容が格納されている
End if
```

システム変数およびセット

コマンドが正しく実行され、有効なドキュメントが選択されると、システム変数OKに1が代入されます。システム変数Documentは選択されたファイルの完全なアクセスパスを格納します。

ファイルが何も選択されていない場合(例えば、ファイルを開くダイアログボックスで**Cancel**ボタンをクリックした場合) 、システム変数OKに0が代入され、システム変数Documentは空になります。

Select folder

Select folder ({message }{;}{ defaultPath {; options}}) -> 戻り値

引数	型		説明
message	文字	➡	ウィンドウのタイトル
defaultPath	文字, 倍長整数	➡	デフォルトのパス名、または、デフォルトのユーザフォルダを表示する空の文字列 (Windowsでは"My documents"、Mac OSでは"Documents")、または記憶されたパス名の番号
options	倍長整数	➡	Mac OS上での選択オプション
戻り値	文字	⮐	選択されたフォルダへのアクセスパス

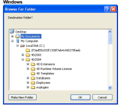
説明

Select folderコマンドは、フォルダを選択するダイアログボックスを表示し、フォルダへの完全なアクセスパスを検索します。オプションの引数defaultPathを使用して、フォルダの場所を指定します。そのフォルダは最初にフォルダ選択ダイアログボックスに表示されます。

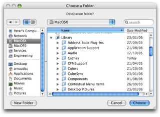
Note: このコマンドは、4Dのカレントフォルダを変更しません。

Select folderコマンドはワークステーションのボリュームおよびフォルダ内をナビゲートするための標準のダイアログボックスを表示します。
オプションの引数messageを指定すると、ダイアログボックス内に表示されます。以下の例では "Select a destination folder" を表示しています。

Windows


Browse For Folder
Destination folder?
Desktop
My Documents
My Computer
Local Disk (C:)
2f3ad55c033f1338f7eb4c46f270faeb
4D2003
4D2004
4D Extensions
4D Runtime Volume License
4D Templates
Databases
Employees
explugins
Make New Folder
OK
Cancel

Mac OS


Choose a Folder
Destination folder?
MacOSX
search
Peter's Compu...
Network
MacOSX
MacOS9
Services
Engineering
Desktop
arnaudsc
Applications
Documents
Movies
Music
Pictures
Name
Date Modified
Library 23/01/06
Address Book Plug-Ins 27/09/03
Application Support 21/08/06
Audio 12/10/05
Caches Today
CFMSupport 21/04/05
Colors 21/10/05
ColorSync 23/01/06
Components 01/08/06
Contextual Menu Items 26/09/03
Desktop Pictures 23/01/06
New Folder
Cancel
Choose

引数defaultPathを使用して、フォルダ選択ダイアログボックスにあるデフォルトフォルダの場所を表示します。3タイプの値をこの引数に渡します:

- カレントプラットフォームのシンタックスを使用している有効なフォルダのパス名。
- システムのデフォルトユーザフォルダを表示する空の文字列("") ("Windowsでは My documents" 、Mac OSでは"Documents") 。
- 記憶されたパス名の番号 (1から32,000) 。それに結び付いているフォルダを表示します。これは、選択ボタンをクリックすると開いているフォルダのパス名が記憶されることを意味します。つまり、ユーザによって選択されたフォルダを指します。このコマンドの最初の呼び出し時に任意の数字 (例えば5) を渡すと、コマンドはシステムのデフォルトユーザフォルダを表示します(空の文字列を渡すことに対応する) 。ユーザはハードディスク上の複数のフォルダをブラウズし、選択ボタンをクリックすると、パス名が番号5に結び付けられます。次に数値の5を渡してこのコ マンドを呼び出すと、記録されたパス名がデフォルトで使用されます。新しい場所が選択されると、5に対応するパスが更新されます。

  このメカニズムを利用して、32,000までのパス名を記憶できます。Windowsではセッションの間のみパスが保持ます。Mac OSでは、一つのセッションから次のセッションまで、パスが保持されています。パス名が正しくないと、引数defaultPathは無視されます。

Note: このメカニズムは、Select documentコマンドで使用しているメカニズムと同じです。記憶されたパス名の数は、この二つのコマンド間で共有されます。

options引数はMac OSにおいて追加の機能を使用できるようにします。この引数に、テーマ中、以下のいずれかの定数を渡すことができます:

定数	型	値	コメント
Package open	倍長整数	2	(Mac OSのみ): パッケージをフォルダとして開きその内容を閲覧できます。この定数が使用されないと、コマンドはパッケージの開封を許可しません。
Use sheet window	倍長整数	16	(Mac OSのみ): 選択ダイアログボックスをシートウィンドウで表示します(Windowsでは、このオプションは無効です)。シートウィンドウはMac OS Xインタフェースに特有なもので、グラフィックアニメーションを伴います (詳細については、DISPLAY SELECTIONを参照してください) 。この定数が使用されないと、コマンドは標準なダイアログボックスを表示します。

1つあるいは両方を合計した値を渡すことができます。このオプションはMac OSでのみ使用できます。Windowsではoptions引数は渡されても無視されます。

ユーザがフォルダを選択して**OK** ボタン (Windows) または**選択** ボタン (Mac OS) をクリックすると、フォルダへのアクセスパスが返されます。

- Windowsでは、アクセスパスは、次のフォーマットで返されます:
  "C:\Folder1\Folder2\SelectedFolder\"

- Mac OSでは、アクセスパスは、次のフォーマットで返されます:
  "Hard Disk:Folder1:Folder2:SelectedFolder:"

Note: Mac OSでは、ダイアログボックス内でフォルダの名前を選択しているかどうかで、返ってくるパスは違うものになる場合があります。

4D Server: この関数は、クライアントワークステーションに接続されているボリュームを見ることができるようにするものです。ストアードプロシージャからこの関数を呼び出すことはできません。

ユーザがダイアログボックスを受け入れるとシステム変数OKは1になります。**Cancel** ボタンをクリックすると、システム変数OKは0になり、関数は空文字を返します。

Note: Windowsでは、ユーザが"ワークステーション" や "ごみ箱" のような何らかの誤った要素を選択すると、ユーザがダイアログボックスを受け入れてもシステム変数OKは0になります。

例題

次の例ではフォルダを選択し、ピクチャライブラリ内のピクチャを保存します。

```
$PictFolder:=Select folder("Select a folder for your pictures.")
PICTURE LIBRARY LIST(pictRefs;pictNames)
For($n;1;Size of array(pictNames))
   $vRef:=Create document($PictFolder+pictNames{$n};"PICT")
   If(OK=1)
      GET PICTURE FROM LIBRARY(pictRefs{$n};$vStoredPict)
      SAVE PICTURE TO FILE($vRef;$vStoredPict)
      CLOSE DOCUMENT($vRef)
   End if
End for
```

SET DOCUMENT CREATOR

SET DOCUMENT CREATOR (document ; fileCreator)

引数	型	説明
document	文字 →	ドキュメント名 または完全なドキュメントのパス名
fileCreator	文字 →	Mac OSファイルクリエータ(4文字の文字列) または空の文字列(Windows)

説明

SET DOCUMENT CREATORコマンドは、documentに渡した名前とパス名を持つドキュメントのクリエータを設定します。

ドキュメントの新しいクリエータをfileCreatorに渡します。

このコマンドは、Windowsでは無効です。

のファイルクリエータについての説明を参照してください。

SET DOCUMENT POSITION

SET DOCUMENT POSITION (DocRef ; offset {; anchor})

引数	型	説明
DocRef	DocRef	⇒ドキュメント参照番号
offset	実数	⇒ファイル位置(バイトで指定)
anchor	倍長整数	⇒1 = ファイル先頭からの相対位置 2 = ファイル最後からの相対位置 3 = 現在位置からの相対位置

説明

このコマンドは、DocRefに渡したドキュメント参照番号を持つ、現在開いているドキュメントだけに機能します。

SET DOCUMENT POSITION コマンドは、引数offsetに渡す、以下の読み込み(RECEIVE PACKET) または書き込み(SEND PACKET) が発生する位置を設定します。

オプションの引数anchorを省略すると、位置はドキュメントの先頭から相対的に表わされます。引数anchorを指定する場合はここにリストされている値のうちいずれかを渡します。

anchorによっては、引数offsetに正の値または負の値を渡すことができます。

SET DOCUMENT PROPERTIES

SET DOCUMENT PROPERTIES (document ; locked ; invisible ; created on ; created at ; modified on ; modified at)

引数	型		説明
document	文字	⇒	ドキュメント名 またはドキュメントの完全なパス名
locked	ブール	⇒	ロックの場合はTrue、アンロックの場合はFalse
invisible	ブール	⇒	非表示の場合はTrue、表示の場合はFalse
created on	日付	⇒	作成日
created at	時間	⇒	作成時間
modified on	日付	⇒	更新日
modified at	時間	⇒	更新時間

説明

SET DOCUMENT PROPERTIESコマンドは、引数document に渡した名前またはパス名を持つドキュメントについての情報を変更します。

呼び出しの前に以下の情報を渡します。

- ドキュメントをロックするには、引数LockedにTrueを渡します。ロックされたドキュメントを開いたり削除することはできません。ドキュメントのロックを解除するにはLockedにFalseを渡します。
- ドキュメントを隠すには、引数invisible にTrueを渡します。デスクトップウィンドウでドキュメントが表示されるようにするには、invisible にFalseを渡します。
- 引数created onおよびcreated atに、ドキュメントの作成日および作成時間を渡します。
- 引数modified onおよびmodified atに、最新のドキュメント更新日および更新時間を渡します。

作成および最新の更新の日付および時間は、ドキュメントを作成、またはこれにアクセスするたびに、システムのファイルマネージャによって管理されます。このコマンドを使用すると、特別な用途のためにこれらのプロパティを変更することができます。GET DOCUMENT PROPERTIESコマンドの例を参照してください。

SET DOCUMENT SIZE

SET DOCUMENT SIZE (DocRef ; size)

引数	型		説明
DocRef	DocRef	→	ドキュメント参照番号
size	実数	→	新しいサイズ(バイト単位)

説明

SET DOCUMENT SIZE コマンドは、ドキュメントのサイズを引数sizeに渡したバイト数に設定します。

ドキュメントが開かれている場合には、引数DocRefにドキュメント参照番号を渡します。

Macintoshでは、ドキュメントデータフォークのサイズが変更されます。

SET DOCUMENT TYPE

SET DOCUMENT TYPE (document ; fileType)

引数	型		説明
document	文字	⇒	ドキュメント名 またはドキュメントの完全なパス名
fileType	文字	⇒	Windowsファイル拡張子 またはMac OSファイルタイプ(4バイト)

説明

SET DOCUMENT TYPEコマンドは、ドキュメントの名前またはパス名を引数documentに渡すドキュメントのタイプを設定します。

引数fileTypeにドキュメントの新しいタイプを渡します。

とDocument Typeの節に記述されているファイルタイプの説明を参照してください。

Windowsでは、このコマンドはファイル拡張子を修正し、その結果documentの値が変わります。

```
SET DOCUMENT TYPE("C:\\Docs\\Invoice.asc";"TEXT")
```

上記のコードを実行すると、"Invoice.asc" ファイルは"Invoice.txt" という名前に変わります。4Dでは、Macintoshの"TEXT" タイプは、Windowsの"txt" タイプに相当します。

4Dに対応するタイプがない場合、拡張子を渡す必要があります。例えば、以下のコードは"Invoice.asc" ファイルを"Invoice.zip" という名前に変えます。

```
SET DOCUMENT TYPE("C:\\Docs\\Invoice.asc";"zip")
```

SHOW ON DISK

SHOW ON DISK (pathname {; *})
pathname 文字 → 表示するアイテムのパス名
* → アイテムがフォルダの場合、その内容を表示

説明

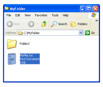
SHOW ON DISKコマンドは、オペレーティングシステムの標準ウィンドウ上に、引数pathname に渡したパス名を持つファイルまたはフォルダを表示します。
ユーザのインターフェースでは、このコマンドを使用して、特定のファイルやフォルダの場所を指定します。

デフォルトで、pathnameがフォルダを指定する場合、コマンドはフォルダ自体のレベルを表示します。任意の引数 * を渡すと、コマンド はフォルダを開き、その内容をウィンドウで表示します。pathname がファイルを指定すると、引数 * は無視されます。

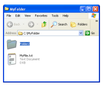
例題

次の例で、このコマンドの機能を表します。

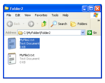
```
SHOW ON DISK("c:\\MyFolder\\MyFile.txt") `指定されたファイルを表示する。
```


MyFolder
File Edit View Favorites Tools Help
Back Search Folders
Address C:\MyFolder Go
Folder2
MyFile.txt
Text Document
0 KB

```
SHOW ON DISK("c:\\MyFolder\\Folder2") `指定されたフォルダを表示する。
```


MyFolder
File Edit View Favorites Tools Help
Back Search Folders
Address C:\MyFolder Go
Folder2
MyFile.txt
Text Document
0 KB

```
SHOW ON DISK("c:\\MyFolder\\Folder2";*) `指定されたフォルダの内容を表示する。
```


Folder2
File Edit View Favorites Tools Help
Back Search Folders
Address C:\MyFolder\Folder2 Go
MyFile2.txt
Text Document
0 KB
MyFile3.txt
Text Document
0 KB

システム変数およびセット

コマンドが正しく実行されると、システム変数OKに1が代入されます。

Test path name

Test path name (pathname) -> 戻り値

引数	型		説明
pathname	文字	→	ディレクトリ、フォルダまたはドキュメントへのパス名
戻り値	倍長整数	←	1=パス名は既存のドキュメントを表す 0=パス名は既存のディレクトリまたはフォルダを表す <0=無効のパス名、OSファイルマネージャエラーコード

説明

Test path nameコマンドは、引数pathnameに渡された名前またはパス名を持つドキュメントまたはフォルダーが、ディスク上に存在するかどうかをチェックします。相対的なパス名または絶対的なパス名のいずれかをカレントシステムのシンタックスで表して渡します。

ドキュメントが見つかれば**Test path name**は1を返します。フォルダーが見つかれば**Test path name**は0を返します。

4Dには、以下のような定義済み定数があります。

定数	型	値
Is a document	倍長整数	1
Is a folder	倍長整数	0

ドキュメントもフォルダも見つからない場合、**Test path name**は負の値を返します。(ファイルが見つからない場合には-43になります) 。

例題

以下の例では、"Journal" というドキュメントがデータベースのフォルダにあるかどうかをテストし、見つからない場合にはこれを作成します。

```
If(Test path name("Journal")#Is a document)
   $vhDocRef:=Create document("Journal")
   If(OK=1)
      CLOSE DOCUMENT($vhDocRef)
   End if
End if
```

TEXT TO DOCUMENT

TEXT TO DOCUMENT (fileName ; text {; charSet {; breakMode}})

引数	型		説明
fileName	文字	→	ドキュメント名またはドキュメントへのパス名
text	テキスト	→	ドキュメントに保存するテキスト
charSet	テキスト, 倍長整数	→	文字コードの名前または数字
breakMode	倍長整数	→	改行の処理モード

説明

TEXT TO DOCUMENTコマンドは、text を直接ディスク上のファイルへと書き込みます。

fileName,引数には書き込みたいファイルへのパス名を渡します。ファイルが存在しない場合には新たに作成されます。このファイルが既にディスク上に存在する場合、以前の内容は消去されます。ただし既に開かれていた場合にはその中身はロックされエラーが静背されます。fileName に渡せるものは以下の通りです:

- ファイル名のみ。例えば "myFile.txt" など。この場合、ファイルはアプリケーションのストラクチャーファイルの隣にある必要があります。
- アプリケーションのストラクチャファイルからの相対パス。例えばWindowsでは "\\docs\\myFile.txt" またはOS Xでは ":docs:myFile.txt"
- 絶対パス。例えばWindowsでは "c:\\app\\docs\\myFile.txt" またはOS Xでは "MacHD:docs:myFile.txt"

ユーザーにドキュメントの名前や場所を指定することを可能にしたい場合は、Open document コマンドまたは Create document コマンドに加え、Documentシステム変数を使用して下さい。

Note: デフォルトでは、このコマンドによって生成されたドキュメントは拡張子を持ちません。拡張子は fileName 内に含める必要があります。SET DOCUMENT TYPEコマンドを使用することもできます。

text 引数には、ディスクに書き込みたいテキストを渡します。文字の定数("my text")を渡す事もできますし、4D テキストフィールドまたは変数を渡す事もできます。

charSet 引数には、ドキュメントに書き込む際の文字コードを渡します。標準の文字コード名(例えば"ISO-8859-1" や "UTF-8")を渡す事もできますし、文字コードの MIBEnum ID (倍長整数)を渡す事もできます。4Dによってサポートされている文字コードの一覧の詳細な情報に関しては、CONVERT FROM TEXT コマンドの詳細を参照して下さい。文字コードに対してバイトオーダーマーク(BOM)が存在している場合、4Dはそれをドキュメント内に挿入します。文字コードを

charSet 引数で指定された文字コードのかわりにバイトオーダーマークが指定した文字コードを使用します(つまり、この引数は無視されます)。文字コードを指定しなかった場合、4Dはデフォルトで "UTF_8"文字コードをBOMとともに使用します。

breakMode 引数には、ドキュメントの改行文字の処理を指示する倍長整数を渡します。"System Documents"テーマ内にある、以下の定数のどれかを渡すことが出きます。

定数	型	値	コメント
Document unchanged	倍長整数	0	何も処理をしません。
Document with CR	倍長整数	3	改行は Mac OSフォーマット(CR、キャリッジリターン)へと変換されます。
Document with CRLF	倍長整数	2	改行はWindowsフォーマット(CRLF、キャリッジリターン＋ラインフィード)へと変換されます。
Document with LF	倍長整数	4	改行はUnixフォーマット(LF、ラインフィード)へと変換されます。
Document with native format	倍長整数	1	改行はOSのネイティブフォーマットに変換されます。Mac OS環境ではCR(キャリッジリターン)に、Windows環境ではCRLF(キャリッジリターン＋ラインフィード)に変換されます。(デフォルトの動作)

デフォルトでは、breakMode 引数を省略した場合、改行はネイティブモード(1)にて処理されます。

Note: このコマンドはOK変数を変更しません。失敗した場合には ON ERR CALL コマンドによって実装されたメソッドを使用することによって割り込み可能なエラーが生成されます。

例題 1

このコマンドの典型的な使用例:

```
TEXT TO DOCUMENT("myTest.txt";"This is a test")
TEXT TO DOCUMENT("myTest.xml";"This is a test")
```

例題 2

ユーザーがファイルを作成する場所を指示できる例:

```
$MyTextVar:="This is a test"
ON ERR CALL("IO ERROR HANDLER")
$vhDocRef :=Create document("")
 // ドキュメントを ".txt" 拡張子で保存
 // この場合、 .txt拡張子は必ずファイル名に追加され、変更することはできません。
If(OK=1) // ドキュメントが正常に作成された場合、
   CLOSE DOCUMENT($vhDocRef) //ドキュメントを閉じる
   TEXT TO DOCUMENT(Document;$MyTextVar )
 // ドキュメントに書き込みが行われます
Else
 // エラー管理
End if
```

VOLUME ATTRIBUTES

VOLUME ATTRIBUTES (volume ; size ; used ; free)

引数	型		説明
volume	文字	→	ボリュームの名前
size	実数	←	ボリュームのサイズ(バイト単位)
used	実数	←	使用サイズ(バイト単位)
free	実数	←	空きサイズ(バイト単位)

説明

VOLUME ATTRIBUTESコマンドは、引数volumeに渡した名前を持つボリュームのサイズ、使用サイズおよび空きサイズをバイト単位で表わして返します。

Note: volumeがマウントされていないリモートボリュームを示す場合、OK変数に0が代入され、3つの引数は-1を返します。

例題

このアプリケーションには、夜間や週末に実行し、ディスク上に大規模な中間ファイルを格納するバッチ処理がいくつか含まれています。このプロセスをできる限り自動的かつ柔軟にするには、中間ファイルを格納するために十分な空きサイズを持つ最初のボリュームを自動的に見つけるルーチンを作成します。例えば以下のようなプロジェクトメソッドを作成します。

```
  ` Find volume for space プロジェクトメソッド
  ` Find volume for space ( 実数 ) -> 文字列
  ` Find volume for space ( 必要なサイズ ) -> ボリューム名または空の文字列

 C_STRING(31;$0)
 C_STRING(255;$vsDocName)
 C_LONGINT($vlNbVolumes;$vlVolume)
 C_REAL($1;$vlSize;$vlUsed;$vlFree)
 C_TIME($vhDocRef)

  `戻り値を初期化する
 $0:=""
  `エラー阻止メソッドを使用して、すべてのI/O処理を保護する
 ON ERR CALL("ERROR METHOD")
  `ボリュームのリストを取得する
 ARRAY STRING(31;$asVolumes;0)
 gError:=0
 VOLUME LIST($asVolumes)
 If(gError=0)
  `Windowsで実行している場合、(通常の) 2つのフロッピーディスクドライブは省略する
    If(On Windows)
       $vlVolume:=Find in array($asVolumes;"A:\\")
       If($vlVolume>0)
          DELETE FROM ARRAY($asVolumes;$vlVolume)
       End if
       $vlVolume:=Find in array($asVolumes;"B:\\")
       If($vlVolume>0)
          DELETE FROM ARRAY($asVolumes;$vlVolume)
       End if
    End if
    $vlNbVolumes:=Size of array($asVolumes)
  `それぞれのボリュームに対して
    For($vlVolume;1;$vlNbVolumes)
  `サイズ、使用サイズ、空きサイズを取得する
       gError:=0
       VOLUME ATTRIBUTES($asVolumes{$vlVolume};$vlSize;$vlUsed;$vlFree)
       If(gError=0)
  `空きサイズは十分にありますか (必要なサイズに32KBを加えたサイズ) ?
          If($vlFree>=($1+32768))
  `もし十分であれば、ボリュームがアンロックされているかどうかをチェック...
             $vsDocName:=$asVolumes{$vlVolume}+Char(Directory symbol)+"XYZ"+String(Random)+".TXT"
             $vhDocRef:=Create document($vsDocName)
             If(OK=1)
                CLOSE DOCUMENT($vhDocRef)
                DELETE DOCUMENT($vsDocName)
  `すべて問題がなければ、ボリュームの名前を返す
                $0:=$asVolumes{$vlVolume}
                $vlVolume:=$vlNbVolumes+1
             End if
          End if
       End if
    End for
 End if
 ON ERR CALL("")
```

このプロジェクトメソッドがアプリケーションに追加されると、例えば、以下のように記述することができます。

```
 $vsVolume:=Find volume for space(100*1024*1024)
 If($vsVolume#"")
  ` Continue
 Else
    ALERT("A volume with at least 100 MB of free space is required!")
 End if
```

VOLUME LIST

VOLUME LIST (volumes)

引数	型		説明
volumes	文字配列	←	現在マウントされているボリュームの名前

説明

VOLUME LISTコマンドは、テキスト配列のvolumes に、現在コンピュータに定義 (Windowsの場合) またはマウント (Mac OSの場合) されているボリュームの名前を代入します。

- Macintoshでは、Finderレベルに表示されるボリュームのリストが返されます。ボリュームの名前のみが返されます (例: "Macintosh HD"や"BootCamp"等)。
- Windowsでは、それぞれのボリュームが物理的に存在しているかどうかに関係なく、現在定義されているボリュームのリストを返します (つまり、CDやDVDが実際にドライブに入っているかどうかに関係なく、ボリュームE:\が返されます) 。ボリューム名はフォルダー区切り文字付きで返されます ("C:\")。

例題

volumes_tテキスト配列に、コンピュータに定義またはマウントされているボリュームのリストを格納するには、以下のように記述します。

```
C_LONGINT($event_l)
$event_l:=Form event
Case of
   :($event_l=On Load)
      ARRAY TEXT(volumes_t;0)
      VOLUME LIST(volumes_t)

   ...
End case
```

Document type

Document type (document) -> 戻り値

引数	型		説明
document	文字	→	ドキュメント名
戻り値	文字	↩	Windowsファイル拡張子(3文字以内の文字列) またはMac OSファイルタイプ (4文字の文字列)

互換性に関するメモ

Mac OS Xでは、Mac OSのファイルタイプは、もうはやサポートされていません。現在では、Windowsのように、名前の接尾辞に基づいてファイルの識別を行います(を参照) 。互換性の理由により、それでも指定された場合は、ドキュメントのMac OSタイプを読み込みます。

説明

Document typeコマンドは、documentに渡したドキュメントの名前とパス名を持つドキュメントのタイプを返します。

Windowsでは、Document typeは、ドキュメントのファイル拡張子(例えば、Microsoft Wordドキュメントを意味する'DOC'や実行ファイルを意味する'EXE'など) 、または対応するMac OSに基づいた4文字のファイルタイプを返します。後者は4DによりまたはMAP FILE TYPESの呼び出しによって、その対等なWindowsのファイル拡張子でマップされた場合です (例えば、'TXT'ファイルの拡張子を意味する'TEXT') 。

Macintoshでは、指定されていると、Document type はドキュメントの4文字のファイルタイプを返します(例えば、テキストドキュメントを意味する'TEXT' 、ダブルクリック可能なアプリケーションを意味する'APPL' など) 。

システム環境

- Count screens
- Current machine
- Current machine owner
- FONT LIST Updated 14.0
- Gestalt
- GET SYSTEM FORMAT
- LOG EVENT
- Menu bar height
- Menu bar screen
- OPEN COLOR PICKER New 14.0
- OPEN FONT PICKER New 14.0
- PLATFORM PROPERTIES
- SCREEN COORDINATES
- SCREEN DEPTH
- Screen height
- Screen width
- Select RGB Color
- SET RECENT FONTS New 14.0
- SET SCREEN DEPTH
- System folder
- Temporary folder
- Font name
- Font number

Count screens

Count screens -> 戻り値

引数	型		説明
戻り値	倍長整数	←	モニターの数

説明

Count screensコマンドは、マシンに接続されている画面モニターの数を返します。

Current machine

Current machine -> 戻り値

引数	型	説明
戻り値	文字	➲ マシンのネットワークの名前

説明

Current machineコマンドは、オペレーティングシステムのネットワークパラメタに設定されたマシンの名前を返します。

例題

クライアント/サーバのバージョンの4D環境で実行していない場合でも、アプリケーションに含まれている幾つかのネットワークサービスが、マシンが正しく構成されていることを必要とする可能性があります。アプリケーションのでは、以下のように記述します。

```
If((Current machine="")|(Current machine owner=""))
  `マシンのネットワークIDを設定するようユーザに要求するダイアログボックスを表示する。
End if
```

Current machine owner

Current machine owner -> 戻り値

引数	型		説明
戻り値	文字	←	マシンオーナーのネットワーク名

説明

Current machine ownerコマンドは、マシンのカレントユーザアカウントで設定されたマシンのオーナー名を返します。

例題

コマンドCurrent machineの例を参照してください。

FONT LIST

FONT LIST (fonts {; listType | *})

引数	型		説明
fonts	テキスト配列	←	フォント名の配列
listType | *	倍長整数, 演算子	→	取得したいフォント型のリスト、フォント名を取得するために*を指定(OS Xのみ)

説明

FONT LIST コマンドは、文字列またはテキスト配列の引数fontsに、システム上で使用可能なフォントの名前を格納します。

listType 引数は取得したいフォントリストの型を指定します。指定するためには、"Font Type List"テーマ内の以下の定数のいずれかを listType 引数に渡して下さい:

定数	型	値	コメント
Favorite fonts	倍長整数	1	fonts にはお気に入りのフォント(マシン上で最もよく使われているフォント)のリストが返されます。 - Windows 環境下では、Windowsコントロールパネル内のアクティブなフォントファミリー名のリストが表示されます。 - OS X 環境下では、コントロールパネル内の "com.apple.Favorites" コレクションのフォントファミリー名のリストが表示されます。英語では"Favorites" 、フランス語では "Favoris" 、ドイツ語では"Favoriten" というように名前がついています。このコレクションは、ユーザーがお気に入りのフォントを何も追加していない場合には空であることがあります。
Recent fonts	倍長整数	2	
System fonts	倍長整数	0	

OS X環境下では、任意の * 引数を渡す事によって fonts 配列を作成し、フォントファミリーの名前ではなくフォント自身の名前を入れます。デフォルトの操作ではリッチテキストエリアのプログラムの管理を簡略化し、フォントファミリーを使用します。つまり * 引数を渡すと、"Arial"、"Arial black" や "Arial narrow" といったフォントファミリー名ではなく、"Arial bold"、"Arial italic"、"Arial narrow italic" といったフォント名が返されるようになります。Windows 環境下では、 * 引数は何の効力も持ちません。渡したとしてもコマンドはフォントファミリー名を返します。

注: Mac OSで、このコマンドから返される結果を ST SET ATTRIBUTES コマンドで使用する場合、 * 引数を渡してはなりません。

例題 1

フォーム上に、システム上で使用可能なフォントリストを表示するドロップダウンリストを作成したいとします。その場合、以下のようなドロップダウンリストのメソッドを記述します。

```
Case of
   :(Form event=On Load)
      ARRAY TEXT(asFont;0)
      FONT LIST(asFont)
  . ...

End case
```

例題 2

最近使用したフォントのリストを取得したい場合:

```
FONT LIST($arrFonts;Recent fonts)
```

Gestalt

Gestalt (selector ; value) -> 戻り値

引数	型		説明
selector	文字	→	4文字のGestaltセレクタ
value	倍長整数	←	Gestalt値
戻り値	倍長整数	↩	エラーコードの結果

説明

Gestaltコマンドは、ユーザがselectorに渡すセレクタに基づいて、システムハードウェアおよびソフトウェアの特性を示す数値をvalueに返します。

必要な情報が得られると、Gestaltコマンドは戻り値として0を、取得できなかった場合には、エラーコード-5550を返します。セレクタが未知のものであれば、コマンドはエラーコード-5551を返します。

重要: GestaltマネージャはMac OSの一部です。セレクタの幾つかはWindows上でも実現されていますが、このコマンドの有効性はWindows上では限られています。

Gestaltに渡すことができるセレクタについての詳細は、Gestalt マネージャに関するApple社の開発者向けドキュメントを参照してください。
オンラインでのアドレスは、以下の通りです。http://developer.apple.com/documentation/Carbon/Reference/Gestalt_Manager/index.html

例題

Mac OSのバージョン10.4.11を使用している場合、以下のコードは、"You are running system version 0x1049" という警告を表示します。

```
$vlErrCode:=Gestalt("sysv";$vlInfo)
If($vlErrCode=0)
   ALERT("You're running system version "+String($vlInfo;"&x"))
End if
```

GET SYSTEM FORMAT

GET SYSTEM FORMAT (format ; value)

引数	型	説明
format	倍長整数	→ 取得するシステムフォーマット
value	文字	← システムで定義されるフォーマットの値

説明

GET SYSTEM FORMATコマンドは、オペレーティングシステムで定義されている幾つかの領域のパラメタのカレント値を返します。このコマンドを使用して、システムの環境設定に基づいた"自動" カスタムフォーマットを作成できます。

引数format には、値を知りたい引数のタイプを渡します。システムは、結果を文字列として引数valueに直接返します。formatには、"" テーマの以下定数の中から一つを必ず渡します。これらの定数の説明は次のとおりです。

定数	型	値	コメント
Currency symbol	倍長整数	2	通貨記号 (例: " ¥ ")
Date separator	倍長整数	13	日付フォーマットの区切り文字 (例: "/")
Decimal separator	倍長整数	0	小数区切り文字 (例: ".")
Short date day position	倍長整数	15	短日付フォーマットでの日の位置: "1" = 左, "2" = 中央, "3" = 右
Short date month position	倍長整数	16	短日付フォーマットでの月の位置: "1" = 左, "2" = 中央, "3" = 右
Short date year position	倍長整数	17	短日付フォーマットでの年の位置: "1" = 左, "2" = 中央, "3" = 右
System date long pattern	倍長整数	8	"dddd MMMM yyyy"形式に対応する長日付表示フォーマット
System date medium pattern	倍長整数	7	"dddd MMMM yyyy"形式に対応する日付表示フォーマット
System date short pattern	倍長整数	6	"dddd d MMMM yyyy"形式に対応する日付表示フォーマット
System time AM label	倍長整数	18	12時間フォーマット時に午前を示すラベル
System time long pattern	倍長整数	5	"HH:MM:SS"形式に対応する長時間表示フォーマット
System time medium pattern	倍長整数	4	"HH:MM:SS"形式に対応する時間表示フォーマット
System time PM label	倍長整数	19	12時間フォーマット時に午後を示すラベル
System time short pattern	倍長整数	3	"HH:MM:SS"形式に対応する時間表示フォーマット
Thousand separator	倍長整数	1	千の位区切り文字 (例: ",")
Time separator	倍長整数	14	時間フォーマットの区切り文字 (例: ":")

例題

機械で印刷される小切手では不正行為を防ぐために、通常払い戻し額に接頭辞として"*"が付けられます。標準システムで通貨の表示フォーマットが"$ 5,422.33" である場合、小切手用のフォーマットは"$***5432.33"です。千の位の後にはコンマはつかず、$記号と最初の数字の間にスペースは入りません。String関数で使用されるフォーマットは "$*******.**"でなければなりません。プログラミングでこの形式を作成するには、通貨記号と小数点記号を取得する必要があります。

```
GET SYSTEM FORMAT(Currency symbol;$vCurrSymb)
GET SYSTEM FORMAT(Decimal separator;$vDecSep)
$MyFormat:=$vCurrSymb+"*******"+$vDecSep+"**"
$Result:=String(amount;$MyFormat)
```

LOG EVENT

LOG EVENT ({outputType ;} message {; importance})

引数	型		説明
outputType	倍長整数	⇒	メッセージの出力タイプ
message	文字	⇒	メッセージの内容
importance	倍長整数	⇒	メッセージの重要度レベル

説明

LOG EVENTコマンドを使用して、アプリケーションの使用中に発生した内部イベントを記録するためのカスタマイズされたシステムを設定します。

イベントに応じて記録されるカスタム情報を messageに渡します。

オプションの引数outputTypeを使用して、messageによって取得された出力チャネルを指定します。Log Eventsテーマにある以下の定数の一つをこの引数に渡します。

定数	型	値	コメント
Into 4D Commands Log	倍長整数	3	この値は4Dのコマンドログファイルがアクティブである場合、このファイルにmessageの内容を記録するよう4Dに指示します。 4DコマンドログファイルはSET DATABASE PARAMETERコマンド (セレクター34) を使用して有効にできます。 **注:** 4Dのログファイルは、**Logs**フォルダに配置されます。このフォルダーはデータベースのストラクチャーファイルと同階層に作成されます(Get 4D folderコマンドを参照) 。
Into 4D Debug Message	倍長整数	1	この値は4Dにmessageをシステムデバッギング環境へ送るよう指示します。結果はプラットホームにより異なります。 • Mac OSでは、コマンドはメッセージをコンソールへ送ります。 • Windowsでは、コマンドはメッセージをデバッグメッセージとして送ります。このメッセージを読むには、Microsoft Visual StudioまたはDebugViewユーティリティが必要です。(http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx)
Into 4D Diagnostic Log	倍長整数	5	ログファイルが有効である場合に、メッセージを4Dのログファイルに記録するよう指示します。 ログファイルはSET DATABASE PARAMETERコマンド (セレクター79) を使用して有効にできます。
Into 4D Request Log	倍長整数	2	この値は4Dリクエストログがアクティブである場合、このファイルにmessageを記録するよう4Dに指示します。
Into Windows Log Events	倍長整数	0	この値は、4Dにmessage をWindowsの"Log events" へ送るよう指示します。このログは起動しているアプリケーションから送られるメッセージを受け取り保存します。この場合オプションのimportance引数を使用してmessageの重要度を設定できます(後述)。 **Notes:** • この特性を利用するには、Windows Log Eventsサービスが起動していなければなりません。 • Mac OSでは、コマンドはこの出力タイプでは何もしません。

outputType引数を渡さない場合、デフォルトでInto Windows Log Events (0) が使用されます。

引数outputTypeにInto Windows Log Eventsを指定すると、オプションの引数importanceを通してmessageに重要度を付けることができます。ログイベントを読んだり理解する助けになります。重要度には情報、警告とエラーの3つのレベルがあります。
4Dには、前もって定義された以下のような定数があります。これらはLog Eventsカテゴリーに置かれています。

定数	型	値
Error Message	倍長整数	2
Information Message	倍長整数	0
Warning Message	倍長整数	1

importance に何も渡さなかったり、正しくない値を渡すと、デフォルト値(0) が使用されます。

例題

WindowsでデータベースIが開かれた時の情報をログしたい場合は、以下のコードをOn Startupデータベースメソッド内に記述します。

```
LOG EVENT(Into Windows Log Events;"The Invoice database was opened.")
```

データベースが開かれるたびに、重要度レベルが0として、この情報がWindowsログイベントに書き込まれます。

Menu bar height

Menu bar height -> 戻り値

引数	型	説明
戻り値	倍長整数	← メニューバーの高さ(ピクセル単位) (メニューバーが表示されていない場合には0)

説明

Menu bar heightは、メニューバーの高さをピクセル数で返します。

メニューバーが隠されている場合、0が返されます。

Menu bar screen

Menu bar screen -> 戻り値

引数	型		説明
戻り値	倍長整数	⇐	メニューバーが表示されている画面の番号

説明

Menu bar screen はメニューバーが表示されている画面の番号を返します。

Windows note: Windowsでは、Menu bar screenはいつも1を返します。

OPEN COLOR PICKER

OPEN COLOR PICKER {(textOrBackground)}

引数	型		説明
textOrBackground	倍長整数	→	0または省略 = テキストカラーを選択、1 = テキスト背景色を選択

説明

OPEN COLOR PICKERコマンドは、システムのカラーピッカーダイアログボックスを表示させます。

Note: これはWindows環境下ではモーダルなダイアログボックスですが、OS X環境下ではモーダルではありません。

ユーザーが色を選択しダイアログボックスを決定すると、選択した色は、フォーカスがあるオブジェクトの「ピッカーの使用を許可」プロパティにチェックが入っている場合、そのオブジェクトのカレントのテキストセレクションに適用されます(*Design Reference* マニュアルを参照して下さい)。

textOrBackground 引数に0を渡すかこの引数を省略した場合、選択した色はテキストへと適用されます。textOrBackgroundに1を渡した場合、選択した色はテキストの背景に適用されます。

色が変更されると、オブジェクトに対して On After Edit フォームイベントが生成されます。

OPEN FONT PICKER

OPEN FONT PICKER
このコマンドは引数を必要としません

説明

OPEN FONT PICKERコマンドは、システムのフォントピッカーダイアログボックスを表示させます。

Note: これはWindows環境下ではモーダルなダイアログボックスですが、OS X環境下ではモーダルではありません。

ユーザーがフォントまたはスタイルを選択しダイアログボックスを決定すると、選択した変更はフォーカスがあるオブジェクトの「ピッカーの使用を許可」プロパティにチェックが入っている場合、そのオブジェクトのカレントのテキストセレクションに適用されます(*Design Reference* マニュアルを参照して下さい)。

色が変更されると、オブジェクトに対して On After Edit フォームイベントが生成されます。

PLATFORM PROPERTIES

PLATFORM PROPERTIES (platform {; system {; processor {; language}}})

引数	型		説明
platform	倍長整数	⇐	2 = Mac OS、3 = Windows
system	倍長整数	⇐	起動しているバージョンによって異なる
processor	倍長整数	⇐	プロセッサファミリー
language	倍長整数	⇐	使用しているシステムによって異なる

説明

PLATFORM PROPERTIESコマンドは、起動しているオペレーティングシステムのタイプ、バージョンやオペレーティングシステムの言語、マシンにインストールされているプロセッサに関する情報を返します。

PLATFORM PROPERTIES は、環境情報を引数platform、system、processorおよびlanguageに渡します。

platformは使用されているオペレーティングシステムを示します。この引数は、前もって定義されている以下の定数の一つを返します。

定数	型	値
Mac OS	倍長整数	2
Windows	倍長整数	3

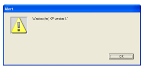
systemに返される情報は、実行している4Dのバージョンによって異なります。

例題

次のプロジェクトメソッドは、使用しているOSソフトウェアを示すアラートボックスを表示します。

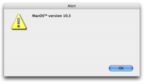
```
  ` SHOW OS VERSION プロジェクトメソッド

 PLATFORM PROPERTIES($vlPlatform;$vlSystem;$vlMachine)
 If(($vlPlatform<2)|($vlPlatform>3))
    $vsPlatformOS:=""
 Else
    If($vlPlatform=Windows)
       $vsPlatformOS:=""
       If($vlSystem<0)
          $winMajVers:=((2^31)+$vlSystem)%256
          $winMinVers:=(((2^31)+$vlSystem)\256)%256
          If($winMinVers=0)
             $vsPlatformOS:="Windows™ 95"
          Else
             $vsPlatformOS:="Windows™ 98"
          End if
       Else
          $winMajVers:=$vlSystem%256
          $winMinVers:=($vlSystem\256)%256
          Case of
             :($winMajVers=4)
                $vsPlatformOS:="Windows™ NT"
             :($winMajVers=5)
                Case of
                   :($winMinVers=0)
                      $vsPlatformOS:="Windows™ 2000"
                   :($winMinVers=1)
                      $vsPlatformOS:="Windows™ XP"
                   :($winMinVers=2)
                      $vsPlatformOS:="Windows™ 2003"
                   Else
                      $vsPlatformOS:="Windows (undetermined version)"
                End case
             :($winMajVers=6)
                Case of
                   :($winMinVers=0)
                      $vsPlatformOS:="Windows™ Vista"
                   :($winMinVers=1)
                      $vsPlatformOS:="Windows™ Seven"
                   Else
                      $vsPlatformOS:="Windows (undetermined version)"
                End case
          End case
       End if
       $vsPlatformOS:=$vsPlatformOS+" version "+String($winMajVers)+"."+String($winMinVers)
    Else
       $vsPlatformOS:="Mac OS™ version "
       If(($vlSystem\256)=16)
          $vsPlatformOS:=$vsPlatformOS+"10"
       Else
          $vsPlatformOS:=$vsPlatformOS+String($vlSystem\256)
       End if
       $vsPlatformOS:=$vsPlatformOS+"."+String(($vlSystem\16)%16)+(("."+String($vlSystem%16))*Num(($vlSystem%16)#0))
    End if
 End if
 ALERT($vsPlatformOS)
```

Windowsでは以下のような警告ボックスが表示されます:



Windows(tm) XP version 5.1

Macintoshでは以下のような警告ボックスが表示されます:



MacOS™ version 10.3

Windowsバージョン

4DのWindowsバージョンが起動している場合、引数systemは32ビット(倍長整数) 値を返します。ビットとバイトは次のように構成されます。

高レベルのバイトに0が代入された場合、Windows NT、Windows 2000、Windows XPまたはWindows Vistaが起動していることを意味します。バイトに1が代入された場合、Windows 95またはWindows 98 (両方とも廃版) が起動していることを意味します。

Note: 高レベルのバイトは、倍長整数値の記号を決定します。そのため、4DでWindows NT、Windows 2000、Windows XPまたはWindows Vistaが起動していることが確かであれば、値の記号をテストするだけです。を使用することも可能です。

低バイトはWindowsのメジャーなバージョン番号を表します。4が返されると、Windows 95、98または Windows NT 4が起動しています。5が返されると、Windows 2000またはWindows XP (両方の場合、値の記号はNT/2000が起動しているかどうかを示します) が起動しています。6が返されると、Windows Vistaが起動しています。

次の低バイトはWindowsのマイナーバージョン番号を表します。Windows 95が起動していると、0が返されます。

Note: % (モジュロ) と \ (整数除算) 数値演算子またはを使用して、これらの値を抽出できます。

引数processorは、マシンのマイクロプロセッサファミリを示します。2つの値が返されます。これらは定数のフォームで利用可能です。

定数	型	値
Intel Compatible	倍長整数	586
Power PC	倍長整数	406

引数platformとprocessorを組み合わせることによって、使用しているマシンが、"MacIntel" タイプであるかどうかを確実に知ることができます (platform=Mac OSとprocessor=Intel Compatible) 。

引数languageを使用して、データベースを実行しているシステムの現在の言語を調べます。以下は、コードとそれに対応する言語を表したリストです。これらのコードは引数に返されます。

Code	Language
1	Arabic
2	Bulgarian
3	Catalan
4	Chinese
5	Czech
6	Danish
7	German
8	Greek
9	English
10	Spanish
11	Finnish
12	French
13	Hebrew
14	Hungarian
15	Icelandic
16	Italian
17	Japanese
18	Korean
19	Dutch
20	Norwegian
21	Polish
22	Portuguese
24	Romanian
25	Russian
26	Croatian
26	Serbian
27	Slovak
28	Albanian
29	Swedish
30	Thai
31	Turkish
33	Indonesian
34	Ukrainian
35	Belarusian
36	Slovenian
37	Estonian
38	Latvian
39	Lithuanian
41	Farsi
42	Vietnamese
45	Basque
54	Afrikaans
56	Faeroese

Note: コマンドがシステムの言語を識別できない場合、値9(English) が返されます。

Macintoshバージョン

4DのMac OSバージョンが起動している場合、引数systemは32ビット(倍長整数) 値を返します。高レベルのワードは使用されません。低レベルのワードの構成は次のとおりです。

- 高バイトは主要なバージョン番号を含みます。
- 低バイトは2ニブル(各4ビット) で構成されます。高ニブルはメジャーアップデートバージョン番号です。低ニブルはマイナーアップデートバージョンです。その例として、システム9.0.4 は$0904のようにコード化され 、少数値2308を受け取ります。

Note: % (モジュロ) と \ (整数除算) 数値演算子またはを使用して、これらの値を抽出できます。

次のフォーミュラを使用して、Mac OSの主なバージョン番号を調べます。

```
 PLATFORM PROPERTIES($vlPlatform;$vlSystem)
 $vlResult:=$vlSystem\256
  `If $vlResult = 8 --> you are under Mac OS 8.x
  `If $vlResult = 9 --> you are under Mac OS 9.x
  `If $vlResult = 16 --> you are under Mac OS 10.x
```

SCREEN COORDINATES

SCREEN COORDINATES (left ; top ; right ; bottom {; screen})

引数	型	説明
left	倍長整数	← 画面エリアの左端のグローバル座標
top	倍長整数	← 画面エリアの上端のグローバル座標
right	倍長整数	← 画面エリアの右端のグローバル座標
bottom	倍長整数	← 画面エリアの下端のグローバル座標
screen	倍長整数	→ 画面番号、または省略した場合には主画面(メインスクリーン)

説明

SCREEN COORDINATESコマンドは、screenに指定した画面のグローバル座標を引数left、top、rightとbottomに返します。

引数screenを省略した場合、このコマンドは主画面(メインスクリーン) の座標を返します。

SCREEN DEPTH

SCREEN DEPTH (depth ; color {; screen})

引数	型	説明
depth	倍長整数	← 画面の深度 (カラーの数 = 2^深度)
color	倍長整数	← 1 = カラー画面、0 = 白黒またはグレイスケール
screen	倍長整数	→ 画面番号、または省略した場合には主画面(メインスクリーン)

説明

Screen depthコマンドは、モニターについての情報を引数depthとcolorに返します。

画面の深度が引数depthに返されます。画面の深度は、モニター上に表示されるカラーの数を表す2のべき乗の指数です。例えば、モニターが256色 (2^8) に設定されている場合、画面の深度は8になります。

4Dでは、以下の表の定義済み定数が用意されています。

定数	型	値
Black and white	倍長整数	0
Four colors	倍長整数	2
Millions of colors 24 bit	倍長整数	24
Millions of colors 32 bit	倍長整数	32
Sixteen colors	倍長整数	4
Thousands of colors	倍長整数	16
Two fifty six colors	倍長整数	8

モニターがカラーを表示するよう設定されている場合、colorには1が返されます。モニターがグレイスケールを表示するよう設定されている場合、colorには0が返されます。この値は、Macintoshのプラットフォーム上で重要であることに注意してください。

4Dでは、以下の表のように前もって定義された定数が用意されています。

定数	型	値
Is color	倍長整数	1
Is gray scale	倍長整数	0

オプションの引数screenは、得たい情報のモニターを指定します。引数screenを省略すると、コマンドは主画面 (メインスクリーン) の深度を返します。

例題

アプリケーションが多くのカラーグラフィックスを表示するとします。その場合には、データベースのどこかに以下のように記述することができます。

```
SCREEN DEPTH($vlDepth;$vlColor)
If($vlDepth<8)
   ALERT("The forms will look better if the monitor"+" was set to display 256 colors or more.")
End if
```

Screen height

Screen height {(*)} -> 戻り値

引数	型		説明
*	演算子	→	Windows: アプリケーションウィンドウの高さ または*が指定されている場合、画面の高さ Macintosh: メイン画面の高さ
戻り値	倍長整数	←	ピクセル数で表される高さ

説明

Windowsでは、Screen heightは、4Dアプリケーションウィンドウ(MDIウィンドウ) の高さを返します。任意の引数* を指定した場合、Screen heightは画面の高さを返します。

Macintoshでは、Screen height はメイン画面の高さを返します。メイン画面とは、メニューバーがある画面のことです。

Screen width

Screen width {(*)} -> 戻り値

引数	型		説明
*	演算子	→	··Windows: *が指定されている場合、アプリケーションウィンドウの幅、または画面の幅 Macintosh: メイン画面の幅
戻り値	倍長整数	↩	ピクセル数で表される幅

説明

Windowsの場合、Screen width は4Dアプリケーションウィンドウ(MDIウィンドウ) の幅を返します。任意の引数 * を指定した場合、Screen widthは画面の幅を返します。

Macintoshの場合、, Screen width はメイン画面の幅を返します。メイン画面とは、メニューバーのある画面のことです。

Select RGB Color

Select RGB Color {(defaultColor {; message})} -> 戻り値

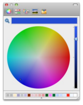
引数	型	説明
defaultColor	倍長整数	→ 事前に選択されたRGBカラー
message	文字	→ 選択ウィンドウのタイトル
戻り値	倍長整数	↩ RGBカラー

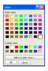
説明

Select RGB Colorコマンドはシステムカラー選択ウィンドウを表示し、ユーザによって選択された色のRGB値を返します。

システムカラー選択ウィンドウは以下のように表示されます。

Macintosh **Windows**





オプションの引数defaultColor を使用して、色を事前に選択できます。例えばこの引数を使用して、ユーザーが最後に設定した色をデフォルトで復元することができます。RGBのフォーマットカラーの値をこの引数に渡します (OBJECT SET RGB COLORSコマンドの説明参照)。SET RGB COLORSテーマにある定数の一つを使用できます。
引数defaultColorを省略したり0を渡すと、ダイアログボックスが開いたときに黒色が選択されます。

オプションの引数messageを使用して、システムウィンドウのタイトルをカスタマイズできます。この引数が省略されると、タイトル"カラー" がデフォルトで表示されます。

このダイアログボックスを受け入れた後の動作はプラットフォームにより異なります:

- Windowsではユーザーが**OK**をクリックすると、コマンドから選択された色がRGBフォーマットで返され、OKシステム変数が1に設定されます。ユーザーがダイアログボックスをキャンセルするとコマンドは-1を返し、システム変数OKに0が設定されます。
- Mac OSではクローズボックスをクリックするか**Esc**キーを押してダイアログボックスを閉じます。ダイアログ内でのユーザーの操作に関わらず、いずれの場合もシステム変数OKは1に設定されます。コマンドは選択された色をRGBフォーマットで返します。ユーザーがカラーを選択しなかった場合、返される値はdefaultColorが渡されていればその値、渡されていなければ0です。

注: サーバーマシンまたはWebプロセス内では、このコマンドを使用しないでください。

SET RECENT FONTS

SET RECENT FONTS (fontsArray)

引数	型		説明
fontsArray	テキスト配列	→	フォント名の配列

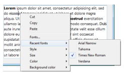
説明

SET RECENT FONTSコマンドは、コンテキストメニュー内の"最近使用したフォント"の中に表示されるフォントの一覧を変更します。

このメニューはセッション中に最近選択されたフォントの名前を表示します。これは特に エリアの中で使用されます。

例題

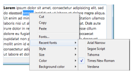
最近使用したフォントの中にフォントを追加したい場合を考えます。


Cut
Copy
Paste
Fonts...
Recent fonts
Style
Size
Color
Background color
Arial Narrow
Tahoma
✓ Times New Roman
Verdana

以下のコードを実行します。

```
ARRAY TEXT($arrRecent;0)
FONT LIST($arrRecent;2)
APPEND TO ARRAY($arrRecent;"Segoe Script")
APPEND TO ARRAY($arrRecent)
```

メニューの中身は以下の様になります。


Cut
Copy
Paste
Fonts...
Recent fonts
Style
Size
Color
Background color
Arial Narrow
Segoe Script
Tahoma
✓ Times New Roman
Verdana

SET SCREEN DEPTH

SET SCREEN DEPTH (depth {; color {; screen}})

引数	型		説明
depth	倍長整数	⇒	画面の深度 (カラーの数 = 2^深度)
color	倍長整数	⇒	1 = カラー画面、0 = グレイスケール
screen	倍長整数	⇒	画面番号、または省略した場合には主画面(メインスクリーン)

説明

SET SCREEN DEPTH は、引数screenに渡した番号を持つ画面の深度とカラー/グレイスケールの設定を変更します。screen引数を省略すると、コマンドは主画面(メインスクリーン) に対して適用されます。

引数colorとdepthに渡す値についての詳細は、SCREEN DEPTHコマンドの説明を参照してください。

System folder

System folder {(type)} -> 戻り値

引数	型		説明
type	倍長整数	→	システムフォルダのタイプ
戻り値	文字	←	システムフォルダへのパス名

説明

System folderコマンドは、アクティブなWindowsまたはMacintoshシステムフォルダにあるシステムフォルダへのパス名、あるいはアクティブなWindowsまたはMacintoshシステムフォルダ自体へのパス名を返します。

オプションの引数typeには、システムフォルダのタイプを示す値を渡します。
System Folderテーマの以下の定義済み定数を使用できます。

定数	型	値	コメント
Applications or Program Files	倍長整数	16	
Desktop	倍長整数	15	
Documents folder	倍長整数	17	OSユーザーのDocumentsフォルダー
Favorites Win	倍長整数	14	
Fonts	倍長整数	1	
Start Menu Win_All	倍長整数	8	
Start Menu Win_User	倍長整数	9	
Startup Win_All	倍長整数	4	
Startup Win_User	倍長整数	5	
System	倍長整数	0	
System Win	倍長整数	12	
System32 Win	倍長整数	13	
User Preferences_All	倍長整数	2	
User Preferences_User	倍長整数	3	

注:

- Windows環境だけでWinキーワードがつけられた定数を使用することができます。それらがMac OS環境で使用されるとき、**System folder**コマンドは空の文字列を返します。
- いくつかのシステム・フォルダへのパス名はログインユーザ(現在のユーザ)のものを返します。定数2～9を用いて、すべてのユーザまたは、ログインユーザのパス名を使用するかを選択することが出来ます。

type引数を省略すると、コマンドはアクティブなシステムフォルダ (定数= System) のパス名を返します。

Temporary folder

Temporary folder -> 戻り値

引数	型		説明
戻り値	文字	←	テンポラリフォルダへのパス名

説明

Temporary folderコマンドは、システムによって設定される現在のテンポラリフォルダへのパス名を返します。

例題

APPEND DATA TO PASTEBOARDコマンドの例を参照してください。

Font name

Font name (fontNumber) -> 戻り値

引数	型	説明
fontNumber	倍長整数	→ フォント名を返すフォント番号
戻り値	文字	↩ フォント名

説明

このコマンドは廃止予定であり、4D v14以降使用されるべきではありません。このコマンドは互換性のために残されていますが今後のバージョンでサポートされなくなります。

Font number

Font number (fontName) -> 戻り値

引数	型		説明
fontName	文字	→	フォント番号を返すフォント名
戻り値	倍長整数	←	フォント番号

説明

このコマンドは廃止予定であり、4D v14以降使用されるべきではありません。このコマンドは互換性のために残されていますが今後のバージョンでサポートされなくなります。

ストラクチャアクセス

ストラクチャアクセスコマンド
CREATE INDEX
DELETE INDEX
Field
Field name
Get external data path
GET FIELD ENTRY PROPERTIES
GET FIELD PROPERTIES
Get last field number
Get last table number
GET MISSING TABLE NAMES
GET RELATION PROPERTIES
GET TABLE PROPERTIES
Is field number valid
Is table number valid
PAUSE INDEXES New 14.0
REGENERATE MISSING TABLE
RELOAD EXTERNAL DATA
RESUME INDEXES New 14.0
SET EXTERNAL DATA PATH
SET INDEX
Table
Table name

ストラクチャアクセスコマンド

この章のコマンドは、データベースストラクチャの情報を返します。これらのコマンドを使用して、テーブル数、各テーブルのフィールド数、テーブルの名前、フィールドの名前、各フィールドのタイプや属性を調べます。ユーティリティコマンドを使用して、迷子のデータを取り戻すために、失われたテーブルを探して再生成することができます。

データベースストラクチャの情報を取得すると、他のデータベースにコピーされる一連のプロジェクトメソッドやフォームを開発する、または使用している場合には非常に便利です。

データベースストラクチャの情報を読み取ることによって、さまざまなデータベースで使用可能になり、移植性の高いコードを生成することができます。

テーブルとフィールドを数える

4Dのバージョン11以降、テーブルとフィールドを削除することができます。従って、以前のバージョンでテーブルとフィールドを数えるために使用していたアルゴリズムを修正する必要があります。現在は、Get last table number、Get last field number、Is table number validそしてIs field number validコマンドを統合したアルゴリズムを使用します。このアルゴリズムの使用例を以下に示します:

```
For($thetable;1;Get last table number)
   If(Is table number valid($thetable))
      For($thefield;1;Get last field number($thetable))
         If(Is field number valid($thetable;$thefield))
            ... `フィールドが存在し有効です。
         End if
      End for
   End if
End for
```

CREATE INDEX

CREATE INDEX (aTable ; fieldsArray ; indexType ; indexName {; *})

引数	型		説明
aTable	テーブル	→	インデックスを作成するためのテーブル
fieldsArray	ポインター配列	→	インデックスされるフィールドへのポインタ
indexType	倍長整数	→	作成されるインデックスのタイプ: -1 = キーワード、0 = デフォルト、1 = 標準 B-Tree、3 = クラスタ B-Tree
indexName	テキスト	→	作成するインデックスの名前
*	演算子	→	渡されると = 非同期インデックス

説明

CREATE INDEXコマンドを使用して以下のインデックスを作成します。

- 1つ以上のフィールドの標準インデックス(複合インデックス) または
- フィールドのキーワードインデックス

fieldsArrayポインタ配列で指定された1つ以上のフィールドを使用して、theTableテーブルのインデックスを作成します。簡単なインデックスを作成する場合、この配列は1行だけ格納します。複合インデックスを作成する場合、この配列は2つ以上の行を格納します (キーワードインデックスの場合は例外) 。複合インデックスでは、インデックスを作成する配列内でのフィールドの順番が重要となります。

indexType引数を使用して、作成されるインデックスのタイプを指定します。Index Typeテーマにある以下の定数のいずれか1つを渡します。

定数	型	値	コメント
Cluster BTree Index	倍長整数	3	クラスタを使用するB-Treeタイプのインデックス。このインデックスタイプは、インデックスが少数のキーを持つ場合、つまり同じ値がデータ内で頻繁に生じる場合に最も適しています。
Default Index Type	倍長整数	0	4Dはフィールドに応じて最適なインデックスのタイプを設定します (キーワードインデックスを除く) 。
Keywords Index	倍長整数	-1	フィールドの内容を単語に分割してインデックス化します。このタイプのインデックスはテキスト、文字、ピクチャー型で使用できます。警告: キーワードを複合タイプにすることはできません。
Standard BTree Index	倍長整数	1	標準 B-Treeタイプのインデックス。この多目的用のインデックスタイプは4Dの以前のバージョンで使用されています。

注: テキスト型のフィールドに設定されたBツリーインデックスは最大で最初の1024文字をインデックス化します。この場合、1024文字以上を含む文字列の検索結果は正しくなりません。

作成するインデックスの名前をindexName引数に渡すことができます。インデックスの命名は、複数の異なるタイプのインデックスを同じフィールドに割り当て、それを個々にDELETE INDEXコマンドで削除する場合に必要となります。indexNameインデックスが既に存在する場合、コマンドは何もしません。

任意の* 引数が渡されると、非同期モードでインデックスを実行します。このモードでは、コマンドからの呼び出し後、インデックスが完了しているか、完了していないかに関わらず元のメソッドがその実行を継続します。

ロックされたレコードがある場合、**CREATE INDEX**コマンドはこれらのレコードをインデックスしません。コマンドはレコードのロックが解除されるのを待ちます。

コマンドを実行している間に問題が発生する場合 (非インデックスフィールド、1つ以上のフィールドでキーワードインデックスを作成する試み等) エラーが発生します。このエラーは、エラー処理メソッドで検知できます。

例題 1

[Customers]テーブルの"Last Name" フィールドと"Telephone" フィールドに標準インデックスをそれぞれ作成。

```
ARRAY POINTER(fieldPtrArr;1)
fieldPtrArr{1}:=->[Customers]LastName
CREATE INDEX([Customers];fieldPtrArr;Standard BTree Index;"CustLNameIdx")
fieldPtrArr{1}:=->[Customers]Telephone
CREATE INDEX([Customers];fieldPtrArr;Standard BTree Index;"CustTelIdx")
```

例題 2

[Customers]テーブルの"Observations"フィールドにキーワードインデックスを作成。

```
ARRAY POINTER(fieldPtrArr;1)
fieldPtrArr{1}:=->[Customers]Observations
CREATE INDEX([Customers];fieldPtrArr;Keywords Index;"CustObsIdx")
```

例題 3

[Customers]テーブルの"City"フィールドと"Zipcode"フィールドに複合インデックスを作成。

```
ARRAY POINTER(fieldPtrArr;2)
fieldPtrArr{1}:=->[Customers]City
fieldPtrArr{2}:=->[Customers]Zipcode
CREATE INDEX([Customers];fieldPtrArr;Standard BTree Index;"CityZip")
```

DELETE INDEX

DELETE INDEX (fieldPtr | indexName {; *})

引数	型		説明
fieldPtr | indexName	ポインター, 文字	⇒	インデックスを削除するフィールドを指すポインタ 削除されるインデックスの名前
*	演算子	⇒	渡されると = 非同期オペレーション

説明

DELETE INDEXコマンドを使用して、データベースから1つ以上の既存のインデックスを削除できます。
フィールドを指すポインター、またはインデックスの名前のどちらかを渡します。

- フィールド(fieldPtr) を指すポインターを渡すと、そのフィールドに関連するすべてインデックスが削除されます。これはキーワードインデックスまたは標準インデックスで構成されます。しかしそのフィールドが1つ以上の複合インデックスに含まれる場合、インデックスは削除できません。この場合、インデックス名を指定しなければなりません。
- インデックス(indexName) の名前を渡すと、指定されたインデックスのみが削除されます。これはキーワードインデックス、標準インデックス、または複合インデックスで構成されます。

任意の* 引数を渡すと、非同期モードでインデックスの削除を実行します。このモードでは、コマンドからの呼び出し後、インデックスの削除が完了しているか、完了していないかに関わらず元のメソッドがその実行を継続します。

fieldPtrまたはindexNameに対応するインデックスがない場合、コマンドは何もしません。

例題

このコマンドの両方のシンタックスの使用例を以下に示します。

```
  `LastNameフィールドに関連するすべてのインデックスを削除
 DELETE INDEX(->[Customers]LastName)
  `"CityZip"という名前のインデックスを削除
 DELETE INDEX("CityZip")
```

Field

Field (aTable ; fieldNum) -> 戻り値

引数	型		説明
aTable	倍長整数	→	テーブル番号
fieldNum	倍長整数	→	フィールド番号
戻り値	ポインター	↩	フィールドポインタ

Field (fieldPtr) -> 戻り値

引数	型		説明
fieldPtr	ポインター	→	フィールドポインタ
戻り値	倍長整数	↩	フィールド番号

説明

Field コマンドには、2つの形式があります。

- tableNumとfieldNumを指定した場合、Fieldはフィールドへのポインタを返します。
- fieldPtrを指定した場合には、Fieldはフィールド番号を返します。

例題 1

以下の例は、変数fieldPtrにテーブル番号=3、フィールド番号=2のフィールドへのポインタを代入します。

```
FieldPtr:=Field(3;2)
```

例題 2

fieldPtr (テーブルの2番目のフィールドを指すポインタ) をField に渡すと、数値2を返します。
以下の例を実行すると変数FieldNumに2を代入します。

```
FieldNum:=Field(FieldPtr)
```

例題 3

以下の例は、変数FieldNumに[Table3]Field2フィールドのフィールド番号を代入します。

```
FieldNum:=Field(->[Table3]Field2)
```

Field name

Field name (fieldPtr | tableNum {; fieldNum}) -> 戻り値

引数	型	説明
fieldPtr | tableNum	ポインター, 倍長整数	→ フィールドポインタ、またはテーブル番号
fieldNum	倍長整数	→ 最初の引数にテーブル番号を渡した場合は、フィールド番号
戻り値	文字	↩ フィールド名

説明

Field name関数は、tableNumとfieldNumまたはfieldPtrで指定したフィールドの名前を返します。

例題 1

以下の例は、FieldArray{1}の2番目の要素にテーブル番号=1、フィールド番号=2のフィールドの名前を代入します。FieldArrayは、2次元の配列です。

```
FieldArray{1}{2}:=Field name(1;2)
```

例題 2

以下の例は、FieldArray{1}の2番目の要素にフィールド[MyTable]MyFieldの名前を代入します。FieldArrayは、2次元の配列です。

```
FieldArray{1}{2}:=Field name(->[MyTable]MyField)
```

例題 3

以下の例は、フィールドに対してポインタを渡して、警告を表示します。

```
ALERT(" テーブル "+Table name(Table($1))+"のフィールド"+Field name($1)+
+"のID番号 は、5文字以上でなければなりません。")
```

Get external data path

Get external data path (aField) -> 戻り値

引数	型		説明
aField	テキスト, BLOB, ピクチャー	→	外部ストレージの場所を取得するフィールド
戻り値	テキスト	←	外部ストレージファイルのフルパス名

説明

Get external data pathコマンドはカレントレコードの、aField引数に渡したフィールドデータの外部ストレージファイルのフルパス名を返します。aField引数にはテキスト、BLOB、またはピクチャー型のフィールドを渡さなくてはなりません。

特にこのコマンドを使用して外部ファイルの再コピーが可能となります。

注: v13の外部ストレージに関しては*Design Reference* マニュアルを参照してください。

このコマンドは以下の場合空の文字列を返します:

- フィールドデータが外部ファイルとして格納されていない場合。
- フィールドがNull値 (外部ファイルが作成されていない) の場合。

GET FIELD ENTRY PROPERTIES

GET FIELD ENTRY PROPERTIES (fieldPtr|tableNum {; fieldNum}; list ; mandatory ; nonEnterable ; nonModifiable)

引数	型	説明
fieldPtr|tableNum	ポインター, 倍長整数	⇒ フィールドポインタ、またはテーブル番号
fieldNum	倍長整数	⇒ 第1引数がテーブル番号の場合、フィールド番号
list	文字	⇐ 関連づけられた選択リストの名前、または空の文字列
mandatory	ブール	⇐ True = 必須入力、False = 任意
nonEnterable	ブール	⇐ True = 表示のみ、False = 入力可
nonModifiable	ブール	⇐ True = 修正不可、False = 修正可

説明

GET FIELD ENTRY PROPERTIESコマンドは、tableNumおよびfieldNum、またはfieldPtrで指定したフィールドのデータ入力プロパティを返します。
次のいずれかの引数を指定することができます。

- tableNumおよびfieldNumに対し、テーブル番号とフィールド番号を指定、または
- fieldPtrにフィールドのポインタを指定

Note: このコマンドは、ストラクチャウインドウレベルで定義したプロパティを返します。同様のプロパティはフォームレベルでも定義できます。

このコマンドが実行されると

- 引数listには、このフィールドに関連付けられた選択リスト名（存在する場合）が返されます。リストは次のタイプのフィールドに関連付けることができます：文字列、テキスト、実数、整数、倍長整数、日付、時間、ブール。
  フィールドに関連付けられた選択リストが存在しない場合や、フィールドタイプが選択リスト用のものでない場合、空の文字列（""）が返されます。
- 引数mandatoryには、フィールドが必須入力であればTrueが、そうでない場合にはFalseが返されます。この必須入力属性は、BLOBを除くあらゆるフィールドタイプに設定することができます。
- 引数nonEnterableには、フィールドが表示のみであればTrueが、そうでない場合にはFalseが返されます。入力不可のフィールドは読み取りのみであり、データの入力はできません。この表示のみ属性は、BLOBを除くあらゆるフィールドタイプに設定することができます。
- 引数nonModifiableには、フィールドが修正不可であればTrueが、そうでない場合にはFalseが返されます。修正不可のフィールドへの入力は一度しか行えず、以後修正はできません。この修正不可属性は、BLOBを除くあらゆるフィールドタイプに設定することができます。

GET FIELD PROPERTIES

GET FIELD PROPERTIES (fieldPtr | tableNum {; fieldNum}; fieldType {; fieldLength {; indexed {; unique {; invisible}}}})

引数	型		説明
fieldPtr | tableNum	ポインター, 倍長整数	→	テーブル番号、またはフィールドポインタ
fieldNum	倍長整数	→	テーブル番号を渡した場合は、フィールド番号
fieldType	倍長整数	←	フィールドのタイプ
fieldLength	倍長整数	←	文字フィールドの場合、長さ
indexed	ブール	←	True = インデックス付き、False = インデックスなし
unique	ブール	←	True = 重複不可、False = 重複あり
invisible	ブール	←	True = 非表示、False = 表示

説明

GET FIELD PROPERTIESコマンドは、tableNumとfieldNumまたはfieldPtrで指定したフィールドの情報を返します。

以下のいずれかの引数を渡します。

- 引数tableNumとfieldNum、または
- fieldPtrにフィールドへのポインタ

コマンドの実行後、以下の情報が返されます。

- fieldTypeにはフィールドのタイプが返されます。以下のような定義済みの定数値が返されます。

定数	型	値
Is Alpha Field	倍長整数	0
Is BLOB	倍長整数	30
Is Boolean	倍長整数	6
Is Date	倍長整数	4
Is Float	倍長整数	35
Is Integer	倍長整数	8
Is Integer 64 bits	倍長整数	25
Is LongInt	倍長整数	9
Is Picture	倍長整数	3
Is Real	倍長整数	1
Is Subtable	倍長整数	7
Is Text	倍長整数	2
Is Time	倍長整数	11

- 引数fieldLenには、フィールドタイプが文字 (つまり、fieldType=Is Alpha Field) の場合、フィールドの長さが返されます。その他のフィールドタイプに対しては、fieldLenの値は意味を持ちません。
- 引数indexedには、フィールドにインデックスが設定されていない場合はFalseが、フィールドにインデックスが設定されている場合はTrueが返されます。indexedの値は、フィールドタイプが文字、整数、倍長整数、実数、日付、時間、ブールの場合にだけ意味を持ちます。
- 引数uniqueには、フィールドが重複不可に設定されているときはTrueが、そうでないときにはFalseが返されます。
- 引数invisibleには、フィールドが非表示に設定されているときにはTrueが、そうでないときにはFalseが返されます。非表示設定は4D標準の(ラベルやチャートなど)エディタで所定のフィールドを隠すために使うことができます。

例題 1

以下の例は、変数vType、vLength、vIndex、vUnique、vInvisibleにテーブル番号=1、フィールド番号=3のフィールドの属性を設定します。

```
GET FIELD PROPERTIES(1;3;vType;vLength;vIndex;vUnique;vInvisible)
```

例題 2

以下の例は、変数vType、vLength、vIndex、vUnique、vInvisibleに[Table3]Field2という名前のフィールドの属性を設定します。

```
GET FIELD PROPERTIES(->[Table3]Field2;vType;vLength;vIndex;vUnique;vInvisible)
```

Get last field number

Get last field number (tableNum | tablePtr) -> 戻り値

引数	型		説明
tableNum | tablePtr	倍長整数, ポインター	→	テーブル番号、またはテーブルポインタ
戻り値	倍長整数	↩	テーブルの最大フィールド番号

説明

Get last field numberコマンドは、tableNumまたはtablePtrにテーブル番号またはポインタを渡したテーブルにあるフィールドの中で、最大のフィールド番号を返します。

フィールドは作成された順に番号が付けられています。フィールドがテーブルから何も削除されていない場合、コマンドはテーブルにあるフィールドの数を返します。テーブルのフィールド番号でループを繰り返す場合は、Is field number validコマンドを使用して、フィールドが削除されているかを確認します。

例題

次のプロジェクトメソッドでは、最初の引数として受け取られたポインタが指すテーブルのフィールド名から成る配列asFieldsを構築します。

```
$vlTable:=Table($1)
ARRAY STRING(31;asFields;Get last field number($vlTable))
For($vlField;Size of array(asFields);1;-1)
   If(Is field number valid($vlTable;$vlField))
      asFields{$vlField}:=Field name($vlTable;$vlField)
   Else
      DELETE FROM ARRAY(asFields;$vlField)
   End if
End for
```

Get last table number

Get last table number -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	データベースの最大テーブル番号

説明

Get last table numberは、データベース中のテーブルの数を返します。テーブルは作成された順番に番号が付けられます。テーブルがデータベースから何も削除されていない場合、コマンドはデータベースにあるテーブルの数を返します。データベースのテーブル番号でループを繰り返す場合は、Is table number validコマンドを使用してテーブルが削除されているかを確認します。

例題

以下の例は、配列asTablesの配列要素を構築します。この配列はフォーム上のドロップダウンリスト(またはタブコントロール、スクロールエリアなど)に割り当てられ、データベース内のテーブルのリストを表示するために使用されます。

```
ARRAY STRING(31;asTables;Get last table number)
If(Get last table number>0) `データベースにテーブルがある場合
   For($vlTables;Size of array(asTables);1;-1)
      If(Is table number valid($vlTables))
         asTables{$vlTables}:=Table name($vlTables)
      Else
         DELETE FROM ARRAY(asTables;$vlTables)
      End if
   End for
End if
```

GET MISSING TABLE NAMES

GET MISSING TABLE NAMES (missingTables)

引数	型		説明
missingTables	テキスト配列	←	データベース中で失われたテーブルの名前

説明

GET MISSING TABLE NAMES コマンドは カレントデータベース中ですべての失われたテーブルの名前をmissingTables配列に返します。

失われたテーブルとは、データファイル中にデータがあるにもかかわらず、カレントストラクチャレベルに存在しないテーブルです。これはデータファイルが異なるバージョンのストラクチャで開かれたときに発生します。

典型的なシナリオは以下の通りです:

- 開発者はテーブルA、B、Cを含むストラクチャを提供する。
- ユーザが (例えば統合されたSQLを使用して) カスタムテーブルDとEを追加し、これらのテーブルにデータを格納する。
- デベロッパが新しいバージョンのストラクチャを提供する。このストラクチャにはテーブルDとEが含まれていません。
  この場合、ユーザバージョンのデータファイルにはテーブルDとEのデータが含まれていますが、アクセスすることはできません。GET MISSING TABLE NAMESコマンドはテーブル名"D"と"E"を返します。

データベースで失われているテーブルを識別したら、新しいREGENERATE MISSING TABLEコマンドを使用して、それらを再アクティブにすることができます。

Note: 失われたテーブルのデータは、テーブルが再生成されていないと、デーファイル圧縮時に失われます。

GET RELATION PROPERTIES

GET RELATION PROPERTIES (fieldPtr|tableNum {; fieldNum}; oneTable ; oneField {; choiceField {; autoOne {; autoMany}}})

引数	型		説明
fieldPtr|tableNum	ポインター, 倍長整数	⇨	フィールドポインタ、またはテーブル番号
fieldNum	倍長整数	⇨	第一引数がテーブル番号の場合、フィールド番号
oneTable	倍長整数	⇦	1テーブルのテーブル番号、または リレーションが未定義の場合は0（ゼロ）
oneField	倍長整数	⇦	1フィールド番号、または リレーションが未定義の場合は0（ゼロ）
choiceField	倍長整数	⇦	選択フィールド番号、または選択フィールドが未定義の場合は0（ゼロ）
autoOne	ブール	⇦	True = 自動1対1リレート False = 1対1マニュアルリレート
autoMany	ブール	⇦	True = 自動1対nリレート False = 1対nマニュアルリレート

説明

GET RELATION PROPERTIESコマンドは、tableNumおよびfieldNum、またはfieldPtrで指定した元のフィールドを起点とするリレート（存在する場合）のプロパティを返します。

次のいずれかの引数を指定することができます。

- tableNumおよびfieldNumに対し、テーブルとフィールドを指定
- fieldPtrにフィールドのポインタを指定

このコマンドが実行されると

- 引数oneTableおよびoneFieldにはそれぞれ、起点フィールドからのリレートが指し示すテーブル番号およびフィールド番号が納められます。このフィールドを起点とするリレートが存在しない場合、これらの引数には0が返されます。
- 引数choicefieldには、このリレートで定義されたターゲットテーブルのワイルドカード選択フィールド番号が納められます。このリレートに対しワイルドカード選択フィールドが設定されていない場合、または起点フィールドからのリレートがない場合、この引数には0が返されます。
- 引数autoOneおよびautoManyにはそれぞれ、このリレートに対し「自動1対1リレート」および「自動1対nリレート」チェックボックスがチェックされている場合にTrueが、そうでない場合にはFalseが返されます。

Note: 引数autoOneおよびautoManyは、起点フィールドから始まるリレートが存在しない場合にもTrueを返します。（この場合、返す値には意味がありません）。2つの引数oneTableおよびoneFieldの値により、リレートが存在するかどうかを確認できます。

GET TABLE PROPERTIES

GET TABLE PROPERTIES (tablePtr| tableNum ; invisible {; trigSaveNew {; trigSaveRec {; trigDelRec {; trigLoadRec}}}})

引数	型		説明
tablePtr| tableNum	ポインター, 倍長整数	→	テーブルポインタ、またはテーブル番号
invisible	ブール	←	True = 非表示、False = 表示
trigSaveNew	ブール	←	True = トリガ"新規レコード保存時"が有効、False = それ以外
trigSaveRec	ブール	←	True = トリガ"既存レコード保存時"が有効、False = それ以外
trigDelRec	ブール	←	True = トリガ"レコード削除時"が有効、False = それ以外
trigLoadRec	ブール	←	***使用しない (廃止) ***

説明

GET TABLE PROPERTIESコマンドは、tablePtrまたはtableNumで渡したテーブルのプロパティを返します。最初の引数としてテーブル番号またはテーブルへのポインタを渡すことができます。

このコマンドが実行されると

- このテーブルに対し非表示属性が設定されている場合、引数invisibleにTrueが返され、そうでない場合Falseが返されます。非表示属性を指定することにより、4D標準のエディタ（ラベル、チャート等）の使用時にテーブルを隠すことができます。
- このテーブルに対し新規レコード保存時トリガが設定されている場合、引数trigSaveNewにTrueが返され、そうでない場合Falseが返されます。
- このテーブルに対し既存レコード保存時トリガが設定されている場合、引数trigSaveRecにTrueが返され、そうでない場合Falseが返されます。
- このテーブルに対しレコード削除時トリガが設定されている場合、引数trigDelRecにTrueが返され、そうでない場合Falseが返されます。

Is field number valid

Is field number valid (tableNum | tablePtr ; fieldNum) -> 戻り値

引数	型		説明
tableNum | tablePtr	倍長整数, ポインター	→	テーブル番号またはテーブルへのポインタ
fieldNum	倍長整数	→	フィールド番号
戻り値	ブール	↩	True = テーブルにあるフィールド False = フィールドがテーブルに存在しない

説明

Is field number validコマンドは、引数fieldNumに渡したフィールド番号のフィールドが、引数tableNumまたはtablePtrに渡したテーブル番号またはポインタのテーブルに存在する場合、Trueを返します。フィールドが存在しない場合、コマンドはFalesを返します。また、フィールドを持つテーブルがエクスプローラーのごみ箱にある場合も、コマンドはFalesを返しますので注意してください。

このコマンドを使用して、フィールドの削除により生じる一連のフィールド番号の欠番を検知することができます。

Is table number valid

Is table number valid (tableNum) -> 戻り値

引数	型		説明
tableNum	倍長整数	→	テーブル番号
戻り値	ブール	←	True = テーブルがデータベースに存在する False = テーブルがデータベースに存在しない

説明

Is table number validコマンドは、引数tableNumに渡したテーブル番号のテーブルがデータベースに存在する場合、Trueを返します。その他の場合は、Falseを返します。また、テーブルがエクスプローラーのごみ箱にある場合、コマンドはFalseを返しますので注意してください。

このコマンドを使用して、フィールドの削除により生じる一連のフィールド番号の欠番を検知することができます。

PAUSE INDEXES

PAUSE INDEXES (aTable)

引数	型		説明
aTable	テーブル	➡	インデックスを停止するテーブル

説明

PAUSE INDEXESコマンドは、aTable のインデックスを、プライマリーキーのインデックスを除き一時的に無効にします。.

インデックスはデータ (.4DIndx file) やデータベースのストラクチャ (_USER_INDEXES,システムテーブル を参照)から削除されるわけではありませんが、無効とされるためそれ以上更新されることはありません。インデックスが 無効化されると、aTable で実行される全てのオペレーション(クエリ、ソート、レコード追加、変更と削除)はインデックスを使用しなくなります。

このコマンドは主に複数のインデックスを持つテーブルのデータを、大量にインポートないし修正する際に有用です。4Dはレコードが検証されるたびにインデックスを更新しなければならないので、このオペレーションは相当な時間を要する可能性があります。インデックスを事前に無効化しておくことでオペレーションの飛躍的にスピードアップさせることが出来ます。

オペレーションが終了した後にインデックスを再開させるためには、aTable に RESUME INDEXES コマンドをしようします。

Note: CREATE INDEX コマンドと DELETE INDEX コマンドを使用することによって似たような結果を得ることが出来ますが、この場合 aTable のそれぞれのインデックスに対してコマンドを呼び出す必要があります。

PAUSE INDEXES コマンドをテーブルに使用し、RESUME INDEXES コマンドをそのテーブルに使用することなくデータベースを閉じた場合、このテーブルのインデックスはデータベースが再開したときに全て自動的にリビルドされます。

Note: このコマンドは4Dリモートからは実行することができません。

例題

大量のデータをインポートするメソッド例:

```
PAUSE INDEXES([Articles])
IMPORT DATA("HugeImport.txt") //Importing
RESUME INDEXES([Articles])
```

REGENERATE MISSING TABLE

REGENERATE MISSING TABLE (tableName)

引数	型		説明
tableName	テキスト	→	再生成する、失われたテーブルの名称

説明

REGENERATE MISSING TABLE コマンドは tableName引数に渡された名前の失われたテーブルを再構築します。失われたテーブルが再構築されると、ストラクチャエディタにそれらが現れ、データに再びアクセスできるようになります。

失われたテーブルとは、データファイル中にデータがあるにもかかわらず、ストラクチャレベルに存在しないテーブルのことです。新しいGET MISSING TABLE NAMESコマンドを使用して、アプリケーション中に存在するかもしれない失われたテーブルを識別できます。

tableNameで指定されたテーブルがデータベースの失われたテーブルでない場合、コマンドはなにも行いません。

例題

以下のメソッドはデータベース中に存在するかもしれないすべての失われたテーブルを再生成します:

```
ARRAY TEXT($arrMissingTables;0)
GET MISSING TABLE NAMES($arrMissingTables)
$SizeArray:=Size of array($arrMissingTables)
If($SizeArray#0)
 // 配列をデータベース中のテーブル名で埋める
  ARRAY TEXT(arrTables;Get last table number)
  If(Get last table number>0) //テーブルが実際に存在すれば
    For($vlTables;Size of array(arrTables);1;-1)
      If(Is table number valid($vlTables))
        arrTables{$vlTables}:=Table name($vlTables)
      Else
        DELETE FROM ARRAY(arrTables;$vlTables)
      End if
    End for
  End if
  For($i;1;$SizeArray)
    If(Find in array(arrTables;$arrMissingTables{$i})=-1)
      CONFIRM("テーブルを再生しますか: "+$arrMissingTables{$i}+"?")
      If(OK=1)
        REGENERATE MISSING TABLE($arrMissingTables{$i})
      End if
    Else
      ALERT("テーブル"+$arrMissingTables{$i}+" を再生できません。データベース中に同じ名前のテーブルがあります。")
    End if
  End for
Else
  ALERT("再生するテーブルがありません。")
End if
```

RELOAD EXTERNAL DATA

RELOAD EXTERNAL DATA (aField)

引数	型		説明
aField	テキスト, BLOB, ピクチャー	→	外部ストレージからリロードを行うフィールド

説明

RELOAD EXTERNAL DATAコマンドはBLOB、ピクチャー、およびテキスト型フィールドに割り当てられた外部ストレージの内容をメモリにリロードします。

このコマンドは、フィールドデータがディスクファイルからメモリにロードされているとき、他のアプリケーションによってディスクファイルが更新されたような場合に使用します (外部ストレージファイルは常に書き込み可能です)。例えばピクチャーフィールドに称されるピクチャーは、ピクチャーエディターで更新し保存することが可能です。

このような場合、以下の2つの理由で**RELOAD EXTERNAL DATA**コマンドを使用してデータを再読み込みしなければなりません:

- フォーム上にフィールドが表示されている場合、その内容を再描画するため。
- レコード保存時に、ディスク上の更新されたピクチャーをメモリ上の古いバージョンで上書きしないため。

注: RELOAD EXTERNAL DATAコマンドは4Dローカルモードおよび4D Server上でのみ動作します。4Dリモートモードで個々にフィールドをリロードすることはできません。4Dリモートモードの場合 (例えばLOAD RECORDコマンドを使用して) レコード全体をリロードしなければなりません。

RESUME INDEXES

RESUME INDEXES (aTable {; *})

引数	型		説明
aTable	テーブル	⇒	インデックスを再開するテーブル
*	演算子	⇒	指定時 = 非同期インデックス

説明

RESUME INDEXES コマンドは、PAUSE INDEXES コマンドを使用して停止させた aTable のインデックスを全て再起動させます。aTable のインデックスが停止されていない場合、コマンドは何もしません。

ほとんどの場合、このコマンドの実行は aTable のインデックスのリビルドをトリガーします。
任意の * 引数を渡した場合、インデックスのリビルドは非同期モードにて実行されます。これはインデックスの完了如何に関わらず、コマンドを呼び出すメソッドは呼び出された後も実行を続けるという事です。この引数を省略した場合、インデックスのリビルドはメソッドの実行をリビルドのオペレーションが完了するまで停止します。

RESUME INDEXES コマンドは4Dサーバーまたはローカルの4Dからのみ呼び出すことが出来ます。このコマンドがリモートの4Dマシンから呼び出されたときは、エラー -10513が生成されます。このエラーは ON ERR CALLコマンドによって実装されたメソッドを使用して割り込むことができます。

SET EXTERNAL DATA PATH

SET EXTERNAL DATA PATH (aField ; path)

引数	型		説明
aField	テキスト, BLOB, ピクチャー	→	ストレージの場所を設定するフィールド
path	テキスト, 倍長整数	→	外部ストレージのパス名およびファイル名、または 0 = ストラクチャー定義を使用する 1 = デフォルトフォルダーを使用する

説明

SET EXTERNAL DATA PATHコマンドはaField引数に渡したフィールドの、カレントレコードの、外部ストレージの場所を設定あるいは変更します。

4Dではテキスト、BLOB、およびピクチャー型のフィールドデータをデータファイルの外部に格納することができます。この機能に関する詳細な説明は*Design Referenceマニュアルを参照してください。*

このコマンドで指定される設定は、カレントレコードがディスクに保存されるときにのみ適用されます。アプリケーションストラクチャーに設定されたパラメーターは変更されません。カレントレコードがキャンセルされると、コマンドはなにも行いません。一度コマンドが実行されると、4Dは自動でレコードフィールドとディスク上のファイルとのリンクを保守します。パスを変更したい場合を除き、再度コマンドを実行する必要はありません。

pathにはカスタムパス名または自動的な場所を指定する定数いずれかを渡すことができます:

- **ファイルへのカスタムパス名**
  この場合外部ストレージをカスタムモードで使用することになります。特定の4Dデータベース機能はこのモードを自動では利用できません (*Design Reference* マニュアル参照)。
  データファイルに対する相対パスあるいは絶対パスを指定できます。パスにはストレージファイルのファイル名と拡張子を含めます。拡張子は実際のデータ型と一致しなければなりません (保存時に自動で変換されることはありません)。システムシンタックスを使用しなければなりません。データベース外部ファイル (*databaseName*.ExternalData) のデフォルトフォルダーを含むどのフォルダーでも指定できます。この場合、これらのファイルはデータベースが保存されるときに含まれます。フォルダーが存在しない場合、4Dが自動で作成します (適切な権限がないなどのため作成に失敗した場合エラーが生成されます)。
  外部ファイルをデータファイルと同階層か、そのサブフォルダーに保存する場合、4Dは指定されたパスがデータファイルに対し相対的であるとみなし、データファイルフォルダーが移動されたり名称変更されたりした場合でもそのリンクを保守します。
  これにより複数のレコードで同じ外部ファイルを共有することが可能な点に留意してください。この外部ファイルに対して行われた変更はすべてのレコードに対して有効です。この場合、複数のプロセスが同時に同じフィールドを変更できるならば、セマフォーを使用して同時アクセスを制限しなければなりません。そうでなければ外部ファイルが損傷するリスクがあります。
- **自動的な場所**
  Data File Maintenanceテーマの、以下の2つの定数を指定できます:

定数	型	値	コメント
Use default folder	倍長整数	1	引数として渡されたフィールドのデータは*databaseName*.ExternalDataという名前のデフォルトフォルダーに保存されます。このフォルダーはデータファイルと同階層に作成されます。このモードでは、外部データを、それがデータファイル内にあるときと同様に、4Dが管理します。
Use structure definition	倍長整数	0	4Dはストラクチャーに設定されたフィールドデータの格納設定を使用します。外部ストレージから内部ストレージに変更しても、外部ファイルは削除されません。

注: SET EXTERNAL DATA PATHコマンドは4Dローカルモードまたは4D Serverでのみ実行できます。リモートモードの4Dではなにも行いません。

例題

ピクチャーデータが特定のサイズ以上の場合、データをデータファイルの外部に保存します。フォームの保存ボタンに以下のように記述します:

```
If(Picture size([Photos]InputField)>1300000)
  // 大きなピクチャーを外部ファイルに保存する
    SET EXTERNAL DATA PATH([Photos]InputField;"C:\\Storage\\LargePicts\\"+String(Record number)+".jpg")
Else
  // 小さなサイズのデータはストラクチャー定義通りに保存する
    SET EXTERNAL DATA PATH([Photos]InputField;Use structure definition)
End if
```

SET INDEX

SET INDEX (aField ; index {; mode} {; *})

aField	フィールド	→	インデックスを作成または削除するフィールド
index	ブール, 整数	→	True=作成、False=削除、または インデックスを作成: -1=キーワード、0=デフォルト、1=標準B-Tree、3=クラスタB-Tree
mode	倍長整数	→	廃止 (引数は無視)
*		→	*を渡すと非同期にインデックスを作成

説明

互換性に関する注意: このコマンドは互換性の目的で保持されています。プログラムでインデックスを管理する場合はCREATE INDEXとDELETE INDEXコマンドの利用を推奨します。

SET INDEXには2つのシンタックスがあります。

- 引数indexにブールを渡す場合、コマンドはaFieldに渡したフィールド用のインデックスを作成したり、削除したりします。
- 引数indexに整数を渡す場合、コマンドは指定されたタイプのインデックスを作成します。

index = ブール
フィールドをインデックスするには、indexにTrueを渡します。コマンドはデフォルトタイプのインデックスを作成します。インデックスが既に存在する場合、呼び出しは機能しません。
indexにFalesを渡すと、コマンドはフィールドに関連づけられたすべての標準インデックス (非複合、非キーワード) を削除します。インデックスが存在しない場合、呼び出しは機能しません。

index = 整数
コマンドはaFieldに対して指定されたインデックスを作成します。FORM LAST PAGEテーマにある以下の定数のいずれか1つを渡します。

定数	型	値	コメント
Cluster BTree Index	倍長整数	3	クラスタを使用するB-Treeタイプのインデックス。このインデックスタイプは、インデックスが少数のキーを持つ場合、つまり同じ値がデータ内で頻繁に生じる場合に最も適しています。
Default Index Type	倍長整数	0	4Dはフィールドに応じて最適なインデックスのタイプを設定します (キーワードインデックスを除く) 。
Keywords Index	倍長整数	-1	キーワードタイプのインデックス。キーワードを複合タイプにすることはできません。fieldsArray配列にはひとつだけフィールドを渡します。
Standard BTree Index	倍長整数	1	標準 B-Treeタイプのインデックス。この多目的用のインデックスタイプは4Dの以前のバージョンで使用されています。

注: テキスト型のフィールドに設定されたBツリーインデックスは最大で最初の1024文字をインデックス化します。この場合、1024文字以上を含む文字列の検索結果は正しくなりません。

SET INDEXコマンドはロックされたレコードのインデックス付けは行いません。レコードがロック解除されるまで待機します。

バージョン11以降、引数modeはもうその役割を果たさなくなっています。渡された場合は無視されます。

オプション引数*を用いて、非同期に(同時に)インデックスが付けられるようになりました。非同期なインデックス付けにより、インデックス付けが終了したかどうかに関係なく、呼び出し側のメソッドの実行をそのまま続行できます。ただし、インデックスが必要なコマンドを実行すると失敗します。

注:

- このコマンドで作成されたインデックスには名前がありません。名前に基づくシンタックスを用いているDELETE INDEXコマンドでそれらのインデックスを削除することはできません。
- 複合インデックスを作成したり削除するために、このコマンドを使用することはできません。
- このコマンドを使用して、CREATE INDEXで作成されたキーワードインデックスを削除することはできません。

例題 1

以下の例は、[Customers]IDフィールドにインデックスを付けます。

```
UNLOAD RECORD([Customers])
SET INDEX([Customers]ID;True)
```

例題 2

[Customers]Name フィールドを非同期モードでインデックスします:

```
SET INDEX([Customers]Name;True;*)
```

例題 3

キーワードインデックスを作成します:

```
SET INDEX([Books]Summary;Keywords Index)
```

Table

Table (tableNum | aPtr) -> 戻り値

引数	型		説明
tableNum | aPtr	倍長整数, ポインター	→	テーブル番号、または テーブルポインタ、または フィールドポインタ
戻り値	ポインター, 倍長整数	↩	テーブル番号を渡した場合テーブルポインタ テーブルポインタを渡した場合テーブル番号 フィールドポインタを渡した場合テーブル番号

説明

Tableコマンドには、3つの形式があります。

- tableNumにテーブル番号を渡した場合は、テーブルポインタを返します。
- aPtrにテーブルポインタを渡した場合は、テーブル番号を返します。
- aPtrにフィールドポインタを渡した場合は、テーブル番号を返します。

例題 1

以下の例は変数tablePtrに3番目のテーブルに対するポインタを代入します。

```
TablePtr:=Table(3)
```

例題 2

変数tablePtr(3番目のテーブルのポインタ) を使用すると、Table関数は数値の3を返します。以下の例を実行すると変数TableNumに3を代入します。

```
TableNum:=Table(TablePtr)
```

例題 3

以下の例は変数tableNumにテーブル[Table3]のテーブル番号を代入します。

```
TableNum:=Table(->[Table3])
```

例題 4

以下の例は変数tableNumにフィールド[Table3]Field1の属するテーブルのテーブル番号を代入します。

```
TableNum:=Table(->[Table3]Field1)
```

Table name

Table name (tableNum | tablePtr) -> 戻り値

引数	型		説明
tableNum | tablePtr	倍長整数, ポインター	→	テーブル番号、またはテーブルポインタ
戻り値	文字	↩	テーブルの名前

説明

Table nameコマンドは、tableNumまたはtablePtrで指定したテーブルの名前を返します。

例題

以下の例は、あるテーブルのレコードを表示します。テーブルへの参照は、テーブルに対するポインタとして渡されます。Table nameコマンドは、ウインドウのタイトルバーにテーブルの名前を表示するために使用されます。

```
  ` SHOW CURRENT SELECTION プロジェクトメソッド
  ` SHOW CURRENT SELECTION ( Pointer )
  ` SHOW CURRENT SELECTION (->[Table])

 SET WINDOW TITLE(Table name($1)+" のレコード")  `ウィンドウタイトル設定
 DISPLAY SELECTION($1->)  ` セレクション表示
```

スペルチェッカー

- Hunspell辞書のサポート
- SPELL ADD TO USER DICTIONARY
- SPELL CHECK TEXT Updated 14.0
- SPELL CHECKING
- SPELL Get current dictionary
- SPELL GET DICTIONARY LIST Updated 14.0
- SPELL SET CURRENT DICTIONARY Updated 14.0

Hunspell辞書のサポート

バージョン12.4および13より、4Dはオープンソースの"Hunspell"辞書をサポートします。この辞書に関する詳細は以下のサイトを参照してください: http://hunspell.sourceforge.net/

4DはMySpellやOpenOffice 2.xと同じフォーマットを使用します。つまり同じ名前の.affと.dicファイルです。例えば"fr-modern"辞書は*fr-modern.aff*と *fr-modern.dic* ファイルで構成されます。

4DアプリケーションでHunspell辞書を使用するには、以下のいずれかの場所に.affと.dicファイルをインストールしなければなりません:

- 4Dアプリケーション内: <4D>/Resources/Spellcheck/Hunspell/
- 4Dデータベース内: <Database_Files>/Resources/Hunspell/

両方の場所とも互換性があります: まずデータベースフォルダーが解析され、次に4Dアプリケーションフォルダー内の辞書が追加されます。同じ名前の辞書が両方の場所にインストールされている場合、データベースフォルダーの辞書が優先されます。この動作により特別な辞書を4Dデータベースフォルダーにカプセル化することができます。

Hunspell辞書は以下からダウンロードできます: http://wiki.services.openoffice.org/wiki/Dictionaries

ユーザーは既存の辞書あるいは新しく作成した辞書を追加できます。辞書を作成する処理はCordialユーザー辞書と同じです (*Design Reference* マニュアルの付録 D: 特別な辞書を使用するを参照)。ユーザー辞書はUTF-8フォーマットで格納します。

SPELL ADD TO USER DICTIONARY

SPELL ADD TO USER DICTIONARY (words)

引数	型		説明
words	テキスト, テキスト配列	⇒	ユーザー辞書に追加する単語または単語リスト

説明

SPELL ADD TO USER DICTIONARYコマンドはカレントのユーザー辞書に単語を追加します。

ユーザー辞書は、ユーザーがカレント辞書に追加した単語を含む辞書のことです。この辞書は*UserDictionaryxxx.dic*という名前のファイルで (xxxはカレント辞書のIDです)、カレントの4Dフォルダーに自動で作成されます。使用される各カレント辞書ごとにユーザー辞書があります。

wordsには、ユーザー辞書に追加する単語を含む文字あるいは文字配列を渡します。すでに登録されている単語は無視されます。

例題

ユーザー辞書に特定の名刺を追加する:

```
ARRAY TEXT($arrTwords;0)
APPEND TO ARRAY($arrTwords;"4D")
APPEND TO ARRAY($arrTwords;"Wakanda")
APPEND TO ARRAY($arrTwords;"Clichy")
SPELL ADD TO USER DICTIONARY($arrTwords)
```

SPELL CHECK TEXT

SPELL CHECK TEXT (text ; errPos ; errLength ; checkPos ; arrSuggest)

引数	型		説明
text	テキスト	→	チェックするテキスト
errPos	倍長整数	←	未知の単語の最初の文字位置
errLength	倍長整数	←	未知の単語の長さ
checkPos	倍長整数	→	チェックを開始する位置
arrSuggest	テキスト配列	←	推奨候補リスト

説明

SPELL CHECK TEXTコマンドはtext引数の内容を、checkPosの位置からチェックし、最初に見つかった未知の単語の位置を返します。

このコマンドはこの未知の単語の最初の文字の位置をerrPosに、そしてその長さをerrLengthに返します。arrSuggest配列には、スペルチェッカーが提案する修正候補のリストが返されます。

スペルチェックがエラーなしで開始され未知の単語が見つかった場合、OKシステム変数に0が設定されます。チェック中に初期化エラーが発生するか未知の単語が見つからなかった場合、OKは1に設定されます。

Note OS X: OS X 環境下でシステムチェッカーが使用されているときには、このコマンドは文法チェックをサポートしません。

例題

テキスト中にあるかもしれないエラーをカウントします:

```
$pos:=1
$errCount:=0
ARRAY TEXT($tErrors;0)
ARRAY TEXT($tSuggestions;0)
Repeat
   SPELL CHECK TEXT($myText;$errPos;$errLength;$pos;$tSuggestions)
   If(OK=0)
      $errCount:=$errCount+1 // エラーカウンター
      $errorWord:=Substring($myText;$errPos;$errLength)
      APPEND TO ARRAY($errors;$errorWord) // エラーの配列
      $pos:=$errPos+$errLength  //チェックを続ける
   End if
Until(OK=1)
 // 最終的に$errCount=Size of array($errorWord)
```

SPELL CHECKING

SPELL CHECKING
このコマンドは引数を必要としません

説明

SPELL CHECKING コマンドは、フィールドまたは現在表示されているフォームでフォーカスを持つ変数のスペルチェックを行います。チェックされるオブジェクトは文字列またはテキストタイプでなければなりません。

スペルチェックは、フィールドまたは変数の最初の文字から始まります。不明な単語が検知されると、スペルチックダイアログボックスが表示されます(詳細については、4DのDesign Referenceマニュアルを参照してください)。SPELL SET CURRENT DICTIONARY コマンドを使用していない限りは、4Dはカレントディクショナリを使用します(アプリケーションの言語に対応しています)。

SPELL Get current dictionary

SPELL Get current dictionary -> 戻り値

引数	型		説明
戻り値	倍長整数	←	スペルチェックに使用される辞書のID

説明

SPELL Get current dictionaryコマンドは使用中の辞書のID番号を返します。

例題

カレント辞書の言語を表示します:

```
  // ロードされた辞書のリスト
SPELL GET DICTIONARY LIST($IDs_al;$Codes_at;$Names_at)
$curLangCode:=SPELL Get current dictionary
$countryName:=$Names_at{Find in array($IDs_al;$curLangCode)}
  // メッセージを表示
ALERT("カレント辞書： "+$countryName) // Spanish
```

SPELL GET DICTIONARY LIST

SPELL GET DICTIONARY LIST (langID ; langFiles ; langNames)

引数	型	説明
langID	倍長整数配列	← 言語のユニークID
langFiles	テキスト配列	← インストールされた言語ファイルの名前
langNames	テキスト配列	← 言語のローカル名

説明

SPELL GET DICTIONARY LISTコマンドはマシンにインストールされたハンスペル辞書ファイルのID、ファイル名、言語名をそれぞれlangID、langFilesそしてlangNames配列に返します。

注: ハンスペル辞書についての詳細な情報は、Hunspell辞書のサポート のセクションを参照して下さい。

- langID には SPELL SET CURRENT DICTIONARY コマンドによって自動的に生成され使用されるID番号が返されます。
IDは固有であり、ファイル名に基づいているという点に注意して下さい。このコマンドは主に開発において有用です。データベースが実行されるたびにIDを再生成する必要がないからです。
- langFiles にはマシンにインストールされた辞書ファイルの名前(拡張子なし)が返されます。
- langNames にはカレントのアプリケーション言語内で指定された言語の名前が返されます。例えば、フランス語の辞書に関してはフランス語を使用するマシンにおいては "français (France)" という値が返され、英語を使用するマシンにおいては "French (France)" と返されます。この言語名の後には "- Hunspell" と続きます。このフィールドは4Dにとって既知のファイルに対してのみ有効です。未知のファイル(例えばカスタムのファイル)に対しては、"N/A - Hunspell" という名前が返されます。(ファイルが有効である限り) これによって辞書の使用が妨げられることはなく、返されたIDはSPELL SET CURRENT DICTIONARY コマンドへと渡す事ができます。

例題

Hunspell辞書に"fr-classic+reform1990.aff"、"fr-classic+reform1990.dic"、"fr-dentist.aff"および"fr-dentist.dic"を配置したとします:

```
ARRAY LONGINT($langID;0)
ARRAY TEXT($dicName;0)
ARRAY TEXT($langDesc;0)
SPELL GET DICTIONARY LIST($langID;$dictName;$langDesc)
```

$langID	$dictName	$langDesc
65536	en_GB	英語 (イギリス)
65792	en_US	英語 (アメリカ合衆国)
131072	de_DE	ドイツ語 (ドイツ)
196608	es_ES	スペイン語
262144	fr_FR	フランス語 (フランス)
1074036166	fr-classic+reform1990	フランス語 (フランス) - Hunspell
1073901273	fr-dentist	No description - Hunspell

SPELL SET CURRENT DICTIONARY

SPELL SET CURRENT DICTIONARY {(dictionary)}

引数	型		説明
dictionary	倍長整数, テキスト	→	スペルチェック用に使用する辞書のID、名前または言語コード 省略した場合デフォルトの辞書を使用

説明

SPELL SET CURRENT DICTIONARYコマンドは、現在の辞書を引数dictionaryに よって指定された辞書で置き換えます。カレント辞書は、4Dに内蔵されているスペルチェック機能に使用されています(詳細については、4D Design Referenceマニュアルや4D Write、4D Viewドキュメントを参照してください) 。設定された現在の辞書は即座にセッションを通して、データベースのすべてのプロセス、および4D Writeや4D Viewプラグインエリアに反映されます。

デフォルトでは、以下の辞書を使用します。

- Windows環境下では、アプリケーションの言語に対応するハンスペル辞書を使用します。
- OS X環境下では、システムのスペルチェッカーを使用します。

Note: ハンスペル辞書の詳細な情報に関しては、Hunspell辞書のサポート のセクションを参照して下さい。

SPELL SET CURRENT DICTIONARY のコマンドを引数なしで呼び出すことによって、デフォルトの辞書をいつでも呼び戻すことができます。

dictionary 引数を使用することによって、辞書を変更することが出来ます。以下のいづれかを渡す事ができます。

- ハンスペルの辞書ID番号(SPELL GET DICTIONARY LIST コマンドによって返されます)。
- ハンスペルの辞書名(ハンスペル辞書のファイル名に対応、拡張子はあってもなくても可)。
- a BCP 47、ISO 639-1 または ISO 639-2 言語コード。例えば、BCP 47 言語コードでは、"en-US" はアメリカ英語を意味し、"en-GB" はイギリス英語を意味します。これらのコードは4D内部で対応するカレントの辞書へとリダイレクトされます(ハンスペルまたはOS Xシステム辞書)。

Compatibility note: 旧バージョンの4Dでは、"コーディアル" 辞書はサポートされていました。互換性のために、"コーディアル"辞書の番号をdictionary 引数に渡す事は可能です(値もしくは "Dictionaries" テーマ内の定数)。しかしながらこの場合、辞書は4D内部でそれに相当するハンスペル辞書(またはOS Xの場合はシステムの辞書)へとリダイレクトされます。

システム変数およびセット

dictionaryが正しくロードされるとシステム変数OKに1が設定されます。そうでなければ0が設定されエラーが返されます。

例題

Hunspellフォルダーに配置した"fr-classic"辞書をロードする:

```
SPELL SET CURRENT DICTIONARY("fr-classic")
 // SPELL SET CURRENT DICTIONARY ("FR-classic.dic") も使用できます
```

セット

- セット
- ADD TO SET
- CLEAR SET
- COPY SET
- CREATE EMPTY SET
- CREATE SET
- CREATE SET FROM ARRAY
- DIFFERENCE
- INTERSECTION
- Is in set
- LOAD SET
- Records in set
- REMOVE FROM SET
- SAVE SET
- UNION
- USE SET

セット

セットはレコードセレクションを素早くパワフルに操作する方法を提供します。作成したセットに対して、カレントセレクションとの関連付け、書き出し、読み込み、消去の処理が行えるだけでなく、4Dは3つの基本機能を提供しています。

- **集合交差** (INTERSECTION)
- **集合結合** (UNION)
- **集合差異** (DIFFERENCE)

セットとカレントセレクション

セットはレコードセレクションを簡潔に表現します。セットの概念はカレントセレクションと密接な関連があります。セットは一般的に以下の目的で使用します:

- 並び順が重要でないセレクションの保存と再利用を実行する場合
- ユーザが画面上で作成したセレクションにアクセスする場合 (UserSet)
- セレクション間で論理演算を実行する場合

カレントセレクションは、現在選択されている各レコードを指し示す参照のリストです。リストはメモリ上に存在します。現在選択されているレコードのみがリストに含まれます。セレクションは実際にレコードを含んでいるわけではなく、レコードに対する参照リストを保持しているだけです。レコードに対する参照はそれぞれ1レコードに対してメモリを4バイト使用します。テーブルに対して作業を実行する場合にも、常にカレントセレクションのレコード用いて作業を行います。セレクションをソートした場合でも、参照のリストがソートされるだけです。カレントセレクションは1つのプロセス内で各テーブルごとに1つしか存在しません。

カレントセレクションと同様に、セットもレコードセレクションを表わします。セットはこれを行うために、非常にコンパクトなレコード参照を使用します。1レコードに対してメモリを1ビット (1/8バイト) 使用します。コンピュータはビットに対する演算を非常に高速に行うため、セットを使った処理は高速に行われます。セットは、セット内にレコードが含まれているかどうかに関係なく、テーブル上に存在するすべてのレコードに対して1ビットずつ使用します。実際、各ビットは1または0であり、この値はレコードがセット内にあるかどうかを表します。

セットはRAMスペースの面から見ると、非常に経済的です。テーブルのレコード件数を8で割れば、そのテーブルのセットサイズをバイト数で求めることができます。例えば、10,000件のレコードを持つテーブルに対してセットを作成すれば、セットはRAMを1,250バイト (約1.2K) 使用します。

各テーブルに対して複数のセットを持つことができます。またデータベースとは別にセットをディスクに保存することもできます。セットに属するレコードを変更するには、最初にセットをカレントセレクションとして使用し、それから1つまたは複数のレコードを修正します。

セットは、ソートした順番には決してなりません。レコードがセットに含まれるか含まれないかだけを示します。これに対して、命名セレクションはソートの順番を保持することができますが、ほとんどの場合より多くのメモリを必要とします。命名セレクションに関する詳細は、を参照してください。

セットは、セットが作成された時点のカレントレコードを"記憶"しています。以下の表は、カレントセレクションとセットの概念を比較したものです:

比較項目	カレントセレクション	セット
1つのテーブルに持てる数	1	0以上
ソート可	はい	いいえ
ディスクに保存可	いいえ	はい
RAM/レコード (バイト)	レコード数 * 4	総レコード数/8
論理演算	いいえ	はい
カレントレコードを記憶	はい	はい (セットが作成された際の)

セットを作成する際、セットはそれを作成したテーブルに属します。セット演算は同じテーブルに属するセット間でのみ可能です。

セットは実在するデータとは別に存在します。これはテーブルを更新をした後では、セットが正確でなくなる可能性があることを意味します。セットが不正確になる可能性のある処理は数多くあります。例えば、すべての東京出身の人でセットを作成した後でその中の1つのレコードを大阪出身に修正しても、セットは更新されません。これは、セットがレコードのセレクションを表現しているに過ぎないためです。レコードを削除した後で新しいレコードを追加した場合やデータの圧縮なども、セットを不正確にします。セットは、その対応するセレクションのデータが更新されていない場合にのみ正確なものであることが保証されます。

プロセスとインタープロセスセット

3種類のセットを使用できます:

- **プロセスセット**: プロセスセットは、それを作成したプロセス内でのみアクセスすることができます。LockedSet もプロセスセットです。プロセスセットはプロセスメソッドが終了しだい消去されます。プロセスセットは、その名前に特別な接頭辞を必要としません。
- **インタープロセスセット**: インタープロセスセットを作成するには、その名前の前に記号 (<>) を付けます。このシンタックスはMac OSとWindows両方で使用できます。
  インタープロセスセットは、データベースのすべてのプロセスからアクセスできます。
  クライアント/サーバーモードで、インタープロセスセットはそれが作成されたマシン上のプロセスから、アクセスできます (クライアントまたはサーバー)。
  インタープロセスセットの名称はデータベース内でユニークでなければなりません。
- **ローカルセット/クライアントセット**: ローカル/クライアントセットはクライアント/サーバモードでの使用を意図しています。ローカル/クライアントセットの名前の先頭にはドル記号 ($) が付けられます。他の種類のセットと異なり、ローカル/クライアントセットはクライアントマシン上に作成されます。

Note: クライアント/サーバモードにおけるセットの使用については4D Server Referenceマニュアルの4D Server: セットと命名セレクションを参照してください。

セットの可視性

以下の表はセットのスコープおよびそれが作成された場所による、セットの可視性についてまとめたものです:

	Client Process	Other client processes	Other clients	Server process	Other server processes
Creation in a client process					
$test	x				
test	x			x (Trigger)	
<>test	x	x			
Creation in a server process					
$test				x	
test				x	
<>test				x	x

セットとトランザクション

トランザクション中でセットを作成できます。トランザクション中で作成されたレコードのセットやトランザクションの外で作成・更新されたレコードの セットを作成できます。トランザクションが終了したとき、トランザクション中に作成されたセットはクリアされるべきです。なぜなら、特にトランザクション がキャンセルされた場合、そのセットはレコードセレクションの表現として正しくないものになっているかもしれないからです。

セットの例題

以下の例では、重複する情報を持つレコードをテーブルから削除します。リストプロパティの設定ルー プはカレントレコードと1つ前のレコードの内容を比較する処理をすべてのレコードに対して行います。名前、住所、郵便番号がすべて一致する場合には、その レコードをセットに追加します。ループが終了したところでセットをカレントセレクションにし、カレントセレクションを削除します:

```
CREATE EMPTY SET([People];"Duplicates")
  ` 重複したレコードを格納する空のセットを作成
ALL RECORDS([People])
  ` 全レコードを選択
  ` ZIP, address, nameで並び替え、重複が前後に来るようにする
ORDER BY([People];[People]ZIP;>;[People]Address;>;[People]Name;>)
  ` 前レコードのフィールド値を保持する変数を初期化
$Name:=[People]Name
$Address:=[People]Address
$ZIP:=[People]ZIP
  ` 先頭と比較するために次レコードに移動
NEXT RECORD([People])
For($i;2;Records in table([People]))
  ` 2から開始してレコード数まで繰り返す
  ` name, address, ZIPが前レコードのそれと同じなら
  ` それは重複レコード
   If(([People]Name=$Name) & ([People]Address=$Address) & ([People]ZIP=$ZIP))
  ` 重複しているカレントレコードをセットに追加
      ADD TO SET([People];"Duplicates")
   Else
  ` 次のレコードと比較するためにname, address, ZIPを保持
      $Name:=[People]Name
      $Address:=[People]Address
      $ZIP:=[People]ZIP
   End if
  ` 次のレコードに移動
   NEXT RECORD([People])
End for
  ` 見つかった重複レコードをセレクションにする
USE SET("Duplicates")
  ` 重複レコードを削除する
DELETE SELECTION([People])
  ` メモリからセットを取り除く
CLEAR SET("Duplicates")
```

メソッドの終りで即座にレコードを削除するのではなく、画面にレコードを表示したり印刷したりして、より詳細な比較を行うこともできます。

UserSetシステムセット

4Dは**UserSet**というシステムセットを維持します。**UserSet**にはユーザによって画面上で最後に選択されたセレクションが自動的に保持されます。したがって、MODIFY SELECTIONやDISPLAY SELECTIONでセレクションを表示し、ユーザにそれから必要なレコードを選択させて、選択結果によるセレクションあるいはセットを作成できます。

4D Server: 名前が"$"で始まっていませんが、**UserSet**システムセットはクライアントセットです。そのため、INTERSECTION、UNION、DIFFERENCEで使用する際は、**UserSet**をクライアントセットと比較していることを確認してください。詳細はこれらのコマンドの説明や、4D Server Referenceマニュアルのの節を参照してください。

UserSetは1つのプロセスに対して1つしかありません。各テーブルごとに**UserSet**があるわけではありません。**UserSet**は、その時点でセレクションを表示しているテーブルに所有されます。

4Dは、デザインモードあるいはMODIFY SELECTIONやDISPLAY SELECTIONコマンドで表示されるリストフォームの**UserSet**を管理します。ただし、このメカニズムはサブフォームには使用されません。

以下のメソッドはレコードを一覧表示し、ユーザにレコードを選択させ、**UserSet**を使用してその選択されたレコードを表示します:

```
  ` 全レコードを表示し、ユーザに必要なだけレコードを選択させる
  ` そしてUserSetを使用して、選択されたレコードをセレクションとする
FORM SET OUTPUT([People];"Display") ` 出力フォームを設定
ALL RECORDS([People]) ` 全レコードを選択
ALERT("Press Ctrl or command and Click to select the people required.")
DISPLAY SELECTION([People]) ` レコードを表示
USE SET("UserSet") ` 選択されたレコードをカレントセレクションにする
ALERT("You chose the following people.")
DISPLAY SELECTION([People]) ` 選択されたレコードを表示
```

LockedSet システムセット

APPLY TO SELECTION、DELETE SELECTION、そしてARRAY TO SELECTIONコマンドは、マルチプロセス環境で使用された場合、LockedSetという名前のセットを作成します。
クエリコマンドもまた、"クエリしてロック"のコンテキストでロックされたレコードを見つけた場合、LockedSet システムセットを作成します (SET QUERY AND LOCK コマンドを参照)。
LockedSet は、コマンドの実行中にロックされたレコードあらわします。

ADD TO SET

ADD TO SET ({aTable ;} set)

引数	型	説明
aTable	テーブル	→ カレントレコードのテーブル, または 省略時デフォルトテーブル
set	文字	→ カレントレコードを追加するセットの名前

説明

ADD TO SETは、setにaTableのカレントレコードを追加します。ここで使用するセットは既に作成されていなければなりません。存在しない場合エラーになります。aTableにカレントレコードが存在しない場合には、ADD TO SETは何も行いません。

CLEAR SET

CLEAR SET (set)

引数	型		説明
set	文字	→	メモリからクリアするセットの名前

説明

CLEAR SETはメモリからsetを消去し、setの占有していたメモリを解放します。CLEAR SETはテーブル、セレクション、レコードには影響を与えません。セットは、消去する前にSAVE SETコマンドを使用して保存することができます。セットはメモリを使用するため、必要のないセットは消去してください。

例題

USE SETの例題参照

COPY SET

COPY SET (srcSet ; dstSet)

引数	型		説明
srcSet	文字	→	コピー元のセット名
dstSet	文字	→	コピー先セット名

説明

COPY SET コマンドは、dstSetセットの中にsrcSetセットの内容をコピーします。

両方のセットともプロセスセット、インタープロセスセット、またはローカルセットが使用できます。

例題 1

以下の例はクライアント/サーバにおいて、クライアントマシン上で管理されるローカルセット"$SetA"をサーバマシン上で管理されるプロセスセット"SetB"にコピーします:

```
COPY SET("$SetA";"SetB")
```

例題 2

以下の例はクライアント/サーバにおいて、サーバマシン上で管理されるプロセスセット"SetA"をクライアントマシン上で管理されるローカルセット"$SetB"にコピーします:

```
COPY SET("SetA";"$SetB")
```

CREATE EMPTY SET

CREATE EMPTY SET ({aTable ;} set)

引数	型		説明
aTable	テーブル	→	空のセットを作成するテーブル, または 省略時、デフォルトテーブル
set	文字	→	新しい空のセットの名前

説明

CREATE EMPTY SETは、aTableに対して新しい空のセットsetを作成します。ADD TO SETコマンドを使って、このセットにレコードを追加できます。既に同じ名前のセットが存在している場合、そのセットを消去して新しい空のセットに置き換えます。

Note: CREATE SETを使用する前に、CREATE EMPTY SETを使用する必要はありません。

例題

の節の例題参照。

CREATE SET

CREATE SET ({aTable ;} set)

引数	型	説明
aTable	テーブル	→ セレクションからセットを作成するテーブル、または 省略時、デフォルトテーブル
set	文字	→ 新規に作成するセットの名前

説明

CREATE SETは、aTableに対して新しいセットsetを作成し、setにカレントセレクションの内容を置きます。テーブルのカレントレコードポインタはsetに保存されます。setに対してUSE SETを使用すると、カレントセレクションとカレントレコードが復元されます。すべてのセットに対してソート順序は適用されません。setが使用されるときはデフォルトの順序が適用されます。既に同じ名前のセットが存在している場合、そのセットを消去し新しいセットに置き換えます。

例題

以下の例は検索を行った後で新しいセットを作成し、それをディスクに保存します:

```
QUERY([People])  ` ユーザが検索を行う
CREATE SET([People];"SearchSet")  ` 新しくセットを作成
SAVE SET("SearchSet";"MySearch")  ` ディスクにセットを保存
```

CREATE SET FROM ARRAY

CREATE SET FROM ARRAY (aTable ; recordsArray {; setName})

引数	型		説明
aTable	テーブル	→	セットのテーブル
recordsArray	倍長整数, ブール配列	→	レコード番号配列、または ブール配列 (True = レコードはセットに含まれる, False = レコードはセットに含まれない)
setName	文字	→	作成するセットの名前, または 省略時、UserSetに適用する

説明

CREATE SET FROM ARRAYコマンドは、セットsetNameを下記の情報から作成します:

- aTableテーブルの絶対レコード番号の配列recordsArray
- ブール配列recordsArray。この場合、配列の値はそれぞれのレコードがsetNameに属する (True) か属さないか (False) を表します。

このコマンドを使用する際、recordsArrayに倍長整数配列を渡すと、配列中のすべての数値はsetNameに格納されるレコードのレコード番号を表します。番号が無効の場合 (例えばレコードが作成されていない場合)、エラー-10503が生成されます。

このコマンドを使用する際、recordsArrayにブール配列を渡すと、配列のN番目の要素は、setNameにN番目のレコードが含まれるか (True) 含まれないか (False) を表します。通常配列の要素数はテーブルのレコード数と一致しなくてはなりません。配列要素数がレコード数より少ない場合、配列により定義されたレコードのみがセットに格納されます。

Note: ブール配列では0からN-1までの要素がこのコマンドにより使用されます。

setName 引数を渡さないか空の文字列を渡すと、コマンドは**Userset**システムセットに適用されます。

エラー管理

倍長整数配列を渡した場合で、レコード番号が無効 (レコードが作成されていない) のとき、エラー-10503が生成されます。

DIFFERENCE

DIFFERENCE (set ; subtractSet ; resultSet)

引数	型		説明
set	文字	⇒	セット
subtractSet	文字	⇒	取り除くセット
resultSet	文字	⇒	結果のセット

説明

DIFFERENCEコマンドは、setとsubtractSetを比較し、subtractSetに格納されている全てのレコードをsetから取り除きます。つまり、setにだけ存在し、subtractSetには存在しないレコードのみをresultSetに格納します。以下の表に、DIFFERENCEコマンドで考えられるすべての組み合わせを示します。

Set1	Set2	結果セット
○	×	○
○	○	×
×	○	×
×	×	×

以下の図に、集合差異演算の処理結果を図で示します。塗りつぶした部分が結果のセットです。

resultSetは、DIFFERENCEコマンドで作成されます。resultSetと同じ名前のセット (set, subtractSetも含めて) が既に存在していた場合、resultSetに置き換えられます。setとsubtractSetは同じテーブルに属していなければなりません。resultSetもsetとsubtractSetと同じテーブルに属します。

4D Server: クライアント/サーバモードにおいて、セットはタイプ (インタープロセス、プロセスおよびローカル) およびそれらがどこで作成されたか (サーバまたはクライアント) によって、アクセス可能かどうかが決定されます。DIFFERENCEでは3つのセットが同じマシン上でアクセスできる必要があります。詳細は4D Server Referenceマニュアルの4D Server: セットと命名セレクションに関する説明を参照してください。

例題

以下の例は、表示したセレクションからユーザが選択したレコードを排除します。このレコードリストは、以下のステートメントで画面に表示されます。

```
DISPLAY SELECTION([Customers]) `customers をリスト表示
```

レコードリストの下部には、オブジェクトメソッド付きのボタンがあります。このオブジェクトメソッドはユーザが選択したレコード (UserSet) を排除し、新しいセットを表示します。

```
CREATE SET([Customers];"$Current") `カレントセレクションのセットを作る
DIFFERENCE("$Current";"UserSet";"$Current") `選択したレコードを除外する
USE SET("$Current") `新しいセットを使用する
CLEAR SET("$Current") `セットを消去する
```

INTERSECTION

INTERSECTION (set1 ; set2 ; resultSet)

引数	型		説明
set1	文字	→	最初のセット
set2	文字	→	2番目のセット
resultSet	文字	→	結果のセット

説明

INTERSECTIONコマンドは、set1とset2を比較し、set1とset2の両方に存在するレコードだけを選択します。下表に、INTERSECTIONコマンドの処理で考えられるすべての組み合わせを示します。

Set1	Set2	Result Set
Yes	No	No
Yes	Yes	Yes
No	Yes	No
No	No	No

以下の図に、集合交差演算の処理結果を図で示します。塗りつぶした部分が結果のセットです。

resultSetはINTERSECTIONコマンドで作成されます。resultSetと同じ名前のセット (set1とset2も含めて) が既に存在する場合はresultSetに置き換わります。set1とset2は同じテーブルに属していなければなりません。resultSetもset1とset2と同じテーブルに属します。set1とset2両方に同じカレントレコードが設定されている場合、そのカレントレコードはresultSetに保持されます。カレントレコードが異なる場合、resultSetはカレントレコードを保持しません。

4D Server: クライアント/サーバモードにおいて、セットはタイプ (インタープロセス、プロセスおよびローカル) およびそれらがどこで作成されたか (サーバまたはクライアント) によって、アクセス可能かどうかが決定されます。INTERSECTIONでは3つのセットが同じマシン上でアクセスできる必要があります。詳細は4D Server Referenceマニュアルの4D Server: セットと命名セレクションに関する説明を参照してください。

例題

以下の例は、"Joe"と"Abby"という2人の販売担当者が重複して担当する顧客を検索します。販売担当者は、各自の顧客を表すセット"Joe"と"Abby"を持っています。

```
INTERSECTION("Joe";"Abby";"Both") ` 両方の顧客のセットを作成する
USE SET("Both") ` セットを使う
CLEAR SET("Both") ` セットを消去、ただし他は残す
DISPLAY SELECTION([Customers]) ` 両方が担当する顧客を表示
```

Is in set

Is in set (set) -> 戻り値

引数	型		説明
set	文字	→	テストするセットの名前
戻り値	ブール	↩	True＝カレントレコードがセットに含まれる False＝カレントレコードがセットに含まれない

説明

Is in set関数は、setの属するテーブルのカレントレコードがsetに含まれているかどうかを調べます。Is in set関数は、カレントレコードがsetに含まれていれば Trueを返し、含まれていなければ Falseを返します。

例題

以下の例は、ボタンのオブジェクトメソッドです。これは、現在表示されているレコードが "Best" のセットに含まれているかどうかを調べます。
:

```
If(Is in set("Best"))  `お得意様かどうか調べる
   ALERT("お得意様"")
Else
   ALERT("お得意様ではありません。")
End if
```

LOAD SET

LOAD SET ({aTable ;} set ; document)

引数	型		説明
aTable	テーブル	→	セットの属しているテーブル、または 省略時、デフォルトテーブル
set	文字	→	作成するセットの名前
document	文字	→	セットを保存したドキュメントの名前

説明

LOAD SETコマンドは、SAVE SETコマンドでディスクに保存したdocumentからセットをメモリに復元します。

documentに格納されたセットは、aTableから作成されてなければいけません。メモリ内に作成されたセットは既に同じセットが存在すると上書きされます。

引数 documentは、セットを保存したドキュメントファイルの名前です。ドキュメントはセットと同じ名前である必要はありません。documentに対して空の文字列を指定すると、"ファイルを開く" ダイアログボックスが表示されます。ユーザはここで復元するセットのファイルを選択できます。

セットは、そのセットが作成された時点のセレクションを表現しているという点に注意して下さい。セットに対応するレコードが更新されると、セットは正確なものでなくなります。したがって、ディスクに保存するセットは、頻繁に更新されることのないレコードで構成するべきです。セットを無効にする操作としては、セットのレコードの修正、セットのレコードの削除、セットを決定した条件の変更等があります。

例題

以下の例は、LOAD SETコマンドを使用して"NY Acme"のセットを復元します。

```
LOAD SET([Companies];"NY Acme";"NYAcmeSt") `セットもメモリにロードします
USE SET("NY Acme") `セレクションをNY Acmeのセットに入れ替えます
CLEAR SET("NY Acme") `セットをメモリから消去します
```

システム変数およびセット

"ファイルを開く" ダイアログボックスで "キャンセル" ボタンをクリックした場合やロード中にエラーが発生した場合はシステム変数OKに0が代入されます。それ以外の場合には1が代入されます。

Records in set

Records in set (set) -> 戻り値

引数	型	説明
set	文字	➡テストするセットの名前
戻り値	倍長整数	↩セットに含まれるレコード数

説明

Records in setコマンドは、setに含まれるレコードの数を返します。setが存在しない場合、またはsetにレコードがない場合には0を返します。

例題

以下の例は、全顧客の中に占めるお得意様の割合をアラートボックスに表示します。

```
  `最初に割合を計算
 $Percent :=(Records in set("Best")/Records in table([Customers]))*100
  `割合を表示する
 ALERT(String($Percent;"##0%")+"がお得意様です。")
```

REMOVE FROM SET

REMOVE FROM SET ({aTable ;} set)

引数	型	説明
aTable	テーブル	→カレントレコードのテーブル、または 省略時、デフォルトテーブル
set	文字	→カレントレコードを取り除くセットの名前

説明

REMOVE FROM SETコマンドは、setからaTableのカレントレコードを取り除きます。セットは存在していなければならず、存在していない場合には、エラーが発生します。テーブルに対するカレントレコードがない場合、REMOVE FROM SETコマンドは何も行いません。

SAVE SET

SAVE SET (set ; document)

引数	型		説明
set	文字	→	保存するセットの名前
document	文字	→	セットを保存するディスクファイルの名前

説明

SAVE SETコマンドは、documentで指定した名前のドキュメントファイルとしてSetをディスクに保存します。

documentは、セットと同じ名前である必要はありません。documentに対して空の文字列 を指定すると、"ファイル作成" ダイアログボックスが表示されます。ユーザは、ここでファイルの名前を入力することができます。保存したセットは、LOAD SETコマンドを使用して復元できます。

"ファイル作成" ダイアログボックスで "キャンセル" ボタンをクリックした場合や、保存処理中にエラーが発生した場合には、システム変数OKに0が代入されます。それ以外の場合には1がセットされます。

SAVE SETコマンドは、時間のかかる検索の結果をディスクに保存するためによく使用されます。

警告: セットは、そのセットが作成された時点のセレクションを表現しているということに注意してください。セットに対応するレコードが更新されると、セットは正確なものではなくなります。したがって、ディスクに保存するセットは、頻繁に更新されることのないレコードで構成するべきです。セットを無効にする操作としては、セットのレコードの修正、セットのレコードの削除、セットを決定した条件の変更等があります。また、セットはフィールド値を保存しないということに注意してください。

例題

以下の例は、ユーザがセットを含んだファイル名を入力するための "ファイル作成" ダイアログボックスを表示します。

```
SAVE SET("SomeSet";"")
```

システム変数およびセット

"ファイル作成" ダイアログボックスで "キャンセル" がクリックされたり、保存処理中にエラーが発生した場合には、システム変数OKに0が代入されます。それ以外の場合には1がセットされます。

UNION

UNION (set1 ; set2 ; resultSet)

引数	型	説明
set1	文字⇒	最初のセット
set2	文字⇒	2番目のセット
resultSet	文字⇒	結果のセット Resulting set

説明

コマンドは、set1とset2のすべてのレコードを含むセットを作成します。下表に、UNIONコマンドの処理で考えられるすべての組み合わせを示します。

Set1	Set2	Result Set
Yes	No	Yes
Yes	Yes	Yes
No	Yes	Yes
No	No	No

以下の図に、集合結合演算の処理結果を図で示します。塗りつぶした箇所が結果セットの部分です。

resultSetは、UNIONコマンドで作成されます。resultSetと同じ名前のセット (set1とset2も含めて) がすでに存在する場合には、resultSetに置き換えられます。set1とset2は同じテーブルに属していなければなりません。resultSetもset1、set2と同じテーブルに属します。resultSetのカレントレコードは、Set1からのカレントレコードです。

4D Server: クライアント/サーバモードにおいて、セットはタイプ (インタープロセス、プロセスおよびローカル) およびそれらがどこで作成されたか (サーバまたはクライアント) によって、アクセス可能かどうかが決定されます。UNIONでは3つのセットが同じマシン上でアクセスできる必要があります。詳細は4D Server Referenceマニュアルの4D Server: セットと命名セレクションに関する説明を参照してください。

例題

この例題では優良顧客のセットにレコードを追加します。二行目のコードでレコードはスクリーンに表示されます。レコードが表示されたのち、優良顧客のセットがディスクからロードされ、ユーザが選択したレコード (セット名 "UserSet") がそのセットに追加されます。最後に新しいセットがディスクに保存されます:

```
ALL RECORDS([Customers]) `全ての顧客を選択
DISPLAY SELECTION([Customers]) `全ての顧客をリスト表示
LOAD SET("$Best";$Path) ` 優良顧客セットをロード
UNION("$Best";"UserSet";"$Best") `任意に選択した顧客をセットに追加する
SAVE SET("$Best";$Path) ` "お得意様" セットをディスクに保存する
```

USE SET

USE SET (set)
引数型　　説明
set　文字→使用するセットの名前

説明

USE SETは、set内のレコードをそのセットの属するテーブルのカレントセレクションにします。

セットを作成すると、その時点のカレントレコードはそのセットに"記憶"されています。USE SETはセット上のカレントレコードの位置を復元し、そのレコードを新しいカレントレコードにします。USE SETを実行する前にこのレコードを削除すると、4Dはセットの先頭のレコードをカレントレコードに設定します。INTERSECTION、UNION、DIFFERENCE、ADD TO SETコマンドはカレントレコードを再設定します。カレントレコードの位置を含まないセットを作成した場合にも、USE SETはセットの先頭のレコードをカレントレコードに設定します。

警告: セットは、そのセットが作成された時点のセレクションを表現しているという点に注意してください。セットに対応するレコードが更新されると、セットは正確なものでないかもしれません。したがって、ディスクに保存するセットは、頻繁に更新されることのないレコードで構成するべきです。セットを無効にする操作としてはセットのレコードの修正、セットのレコードの削除、セットを決定した条件の変更等があります。

例題

以下の例はLOAD SETを使用して、所在地が"NY Acme"のセットを復元し、その復元したセットをUSE SETを使用してカレントセレクションにします:

```
LOAD SET([Companies];"NY Acme";"NYAcmeSt")  ` セットをメモリにロード
USE SET("NY Acme")  ` カレントセレクションをNY Acmeに変更
CLEAR SET("NY Acme")  ` メモリからセットをクリア
```

セレクション

- ALL RECORDS
- APPLY TO SELECTION
- Before selection
- DELETE SELECTION
- DISPLAY SELECTION
- Displayed line number
- End selection
- FIRST RECORD
- GET HIGHLIGHTED RECORDS
- GOTO SELECTED RECORD
- HIGHLIGHT RECORDS
- LAST RECORD
- MODIFY SELECTION
- NEXT RECORD
- ONE RECORD SELECT
- PREVIOUS RECORD
- Records in selection
- REDUCE SELECTION
- SCAN INDEX
- Selected record number
- TRUNCATE TABLE

ALL RECORDS

ALL RECORDS {(aTable)}

引数	型		説明
aTable	テーブル	→	すべてのレコードを選択するテーブル 省略時、デフォルトテーブル

説明

ALL RECORDSは、aTableの全レコードをカレントプロセスのカレントセレクションにします。ALL RECORDSは先頭のレコードをディスクからロードし、カレントレコードに設定します。ALL RECORDSは、レコードの順序をデフォルトのレコード順序に戻します。

例題

以下の例は、[People]テーブルのすべてのレコードを表示します:

```
ALL RECORDS([People]) ` テーブルの全レコードをカレントセレクションにする
DISPLAY SELECTION([People]) ` 出力フォームにレコードを表示
```

APPLY TO SELECTION

APPLY TO SELECTION (aTable ; statement)

引数	型		説明
aTable	テーブル	→	ステートメントを適用するテーブル
statement	命令文	→	1行のコードで記述されたメソッド

説明

APPLY TO SELECTIONは、aTableのカレントセレクションに対してstatementを適用します。statementは1行のステートメントまたはメソッドのどちらでも構いません。statementがaTableのレコードを修正した場合、そのレコードをディスクに保存されます、レコードを修正しない場合には保存しません。カレントセレクションが空の場合、APPLY TO SELECTIONは何も行いません。リレーションが自動であれば、statementはリレート先のテーブルのフィールドを含むことができます。

APPLY TO SELECTIONは、カレントセレクションの情報 (例えば合計等) を求めるため、あるいはセレクション中のレコードを修正するため (例えばフィールドの頭文字を大文字に変える等) に使用します。このコマンドをトランザクション内で使用されている場合、トランザクション処理が取り消されると、すべての変更は無効とされます。

4D Server: statementに渡されるコマンドはサーバでは実行されません。セレクションの各レコードは修正のためローカルのワークステーションに送り返されます。

APPLY TO SELECTIONを実行している間、処理の進捗を表すサーモメータが表示されます。APPLY TO SELECTIONを呼び出す前に、MESSAGES OFFを使用してサーモメータの表示を取り消すことができます。サーモメータが表示されると、ユーザは処理をキャンセルすることができます。

例題 1

以下の例題はテーブル [Employees] 中のカレントセレクションのレコードを大文字に変更します:

```
APPLY TO SELECTION([Employees];[Employees]Last Name:=Uppercase([Employees]Last Name))
```

例題 2

APPLY TO SELECTION実行中にレコードを修正して、そのレコードがロックされていると、更新は保存されません。ロックされたレコードはLockedSetと呼ばれるセットに格納されます。APPLY TO SELECTION実行後、LockedSetをテストしてロックされたレコードがなかったか確認します。以下のループはすべてのレコードが更新されるまで処理を実行します:

```
Repeat
   APPLY TO SELECTION([Employees];[Employees]Last Name:=Uppercase([Employees]Last Name))
   USE SET("LockedSet") ` ロックされたレコードのみを選択
Until(Records in set("LockedSet")=0) ` ロックされたレコードがなくなれば終了
```

例題 3

この例題ではメソッドを使用します:

```
ALL RECORDS([Employees])
APPLY TO SELECTION([Employees];M_Cap)
```

システム変数およびセット

ユーザが進捗サーモメータの停止ボタンをクリックすると、OKシステム変数に0が設定されます。そうでなければ1が設定されます。

Before selection

Before selection {(aTable)} -> 戻り値

引数	型	説明
aTable	テーブル	→ レコードポインタがセレクションの先頭より前に あるかをテストするテーブル, または 省略時、デフォルトテーブル
戻り値	ブール	↩ Yes (TRUE) または No (FALSE)

説明

Before selectionは、カレントレコードポインタがaTableのカレントセレクションの前にある場合にTRUEを返します。Before selectionは、一般にPREVIOUS RECORDにより、カレントレコードポインタが先頭レコードの前に移動したかどうかを調べるために使用します。カレントセレクションが空の場合、Before selectionはTRUEを返します。

カレントレコードポインタをセレクションに内に戻すには、LAST RECORD、FIRST RECORD、GOTO SELECTED RECORDを使用します。NEXT RECORDではポインタはセレクション内に戻りません。

PRINT SELECTIONまたはプリント...メニューを選択してレポートを印刷する場合も、Before selectionは最初のヘッダでTRUEを返します。以下のステートメントを使用して最初のヘッダを判定し、先頭ページに特殊なヘッダを印刷することができます:

```
  ` レポート印刷に使用される出力フォームのメソッド
 $vpFormTable:=Current form table
 Case of
  ` ...
    :(Form event=On Header)
  ` ヘッダエリアが印刷されようとしている
       Case of
          :(Before selection($vpFormTable->))
  ` 最初のブレークヘッダ用のコード
  ` ...
       End case
 End case
```

例題

以下の例はレポートの印刷中に使用します。変数vTitleを設定し、先頭ページのヘッダエリアに印刷します:

```
  ` [Finances];"Summary" フォームメソッド
 Case of
  ` ...
    :(Form event=On Header)
       Case of
          :(Before selection([Finances]))
             vTitle:="Corporate Report 1997" ` 1ページめのタイトル
          Else
             vTitle:="" ` 他のページではタイトルを印刷しない
       End case
 End case
```

DELETE SELECTION

DELETE SELECTION {(aTable)}

引数	型		説明
aTable	テーブル	→	カレントセレクションを削除するテーブル, または 省略時、デフォルトテーブル

説明

DELETE SELECTIONは、aTableのカレントセレクションのレコードを削除します。カレントセレクションが空の場合、**DELETE SELECTION**は何も行いません。レコードが削除された後、カレントセレクションは空になります。トランザクション処理中に削除されたレコードは、トランザクション処理が実行または取り消されるまで、他のユーザや他のプロセスに対してロックされます。

警告: レコードの削除は、恒久的な操作です。削除が実行されると元に戻すことはできません。

テーブルインスペクターの「レコードを完全に削除」オプションの選択を解除すると、**DELETE SELECTION**使用時のレコード削除処理を高速にすることができます (4Dデザインリファレンスマニュアルのレコードを完全に削除参照)。

例題 1

以下の例は[People]テーブルのすべてのレコードを表示します。ユーザはこの中から削除したいものを選択します。この例には2つのメソッドがあります。第1のメソッドはレコードを表示します。第2は削除ボタンのオブジェクトメソッドです。以下は、第1のメソッドです:

```
ALL RECORDS([People])  ` 全レコードをセレクションに
FORM SET OUTPUT([People];"Listing")  ` レコードを一覧するフォームを設定
DISPLAY SELECTION([People])  ` 全レコードを表示
```

次に示すのは削除ボタンのオブジェクトメソッドです。このボタンは出力フォームのフッタエリアにあります。このオブジェクトメソッドでは、セレクションを削除するために、ユーザが選択したレコード (UserSet) を使用します。ユーザがレコードを1件も選択しなかった場合、DELETE SELECTIONは何も行わないという点に注目してください。

```
  ` 本当にレコードを削除するか確認する
CONFIRM("You selected "+String(Records in set("UserSet"))+" people to delete."
+Char(13)+"Click OK to Delete them.")
If(OK=1)
   USE SET("UserSet")  ` ユーザが選択したレコードを使用
   DELETE SELECTION([People])  ` レコードセレクションを削除
End if
ALL RECORDS([People])  ` すべてのレコードを選択
```

例題 2

DELETE SELECTIONの実行中にロックされたレコードが見つかると、そのレコードは削除されません。ロックされたレコードはすべて"**LockedSet**"というセットに納められます。**DELETE SELECTION**の実行後、"**LockedSet**"を調べて、ロックされているレコードが存在していたかどうかを知ることができます。次はループを使用してすべてのレコードを削除します:

```
Repeat  ` ロックされたレコードがある限り繰り返す
   DELETE SELECTION([ThisTable])
   $lockedRecords:=Records in set("LockedSet")
   If($lockedRecords#0)  ` ロックされたレコードがあれば
      USE SET("LockedSet")  ` ロックされたレコードのみをセレクションにする
   End if
Until($lockedRecords=0)  ` ロックされたレコードがなくなるまで
```

DISPLAY SELECTION

DISPLAY SELECTION ({aTable}{; selectMode}{; enterList}{; *}{; *})

aTable	テーブル	→ 表示するテーブル, または 省略時、デフォルトテーブル
selectMode	倍長整数	→ 選択モード
enterList	ブール	→ リスト入力を許可するオプション
*		→ 1レコードセレクションの場合にも出力フォームを使用し、入力フォームのスクロールバーを隠す

説明

DISPLAY SELECTION は、出力フォームを使用して、aTableの カレントセレクションを表示します。レコードはデザインモードの一覧に類似のスクロール可能な一覧として表示されます。ユーザがレコー ドをダブルクリックすると、デフォルトでそのレコードはカレント入力フォーム上に表示されます。リストは最前面にあるウインドウに表示されます。

セレクションを表示し、さらに (デザインモードのウインドウで実行するときにように) レコードをダブルクリックしてカレント入力フォーム上で修正するには、DISPLAY SELECTIONではなくMODIFY SELECTIONを使用してください。
以降に説明する情報はレコードの修正に関する情報を除き、すべて両方のコマンドに適用されます。

DISPLAY SELECTIONを実行した直後、カレントレコードが存在しない場合があります。カレントレコードを必要とする場合は、FIRST RECORDやLAST RECORD等のコマンドを使用してください。

selectModeを使用し、マウスを用いたリスト上のレコードを選択の動作を設定できます。この引数には "Form Parameters" テーマのいずれかの定数を渡すことができます:

定数	型	値	コメント
Multiple Selection	倍長整数	2	複数レコードを同時に選択することができます。連続しているレコードを選択するには、選択する最初のレコードをクリックし、次に**Shift**キーを押しながらセレクションに含めたい最後のレコードをクリックします。連続しないレコードを選択するには、**Ctrl**キー (Windows) または、**Command**キー (Mac OS) を押したまま各レコードをクリックします。
No Selection	倍長整数	0	リスト上のレコードを選択することはできません。
Single Selection	倍長整数	1	一度に1レコードだけを選択できます。

selectModeを渡さない場合は、デフォルトとしてMultiple Selectionモードが使用されます。

enterList引数を使用すると、表示されるリスト中でリスト更新を許可することができます。これにより、ユーザは直接出力フォーム上でレコードを選択し、値を変更できるようになります。このモードを有効にするにはTrueを渡し、無効にするにはFalseを渡します。引数enterListを渡さない場合、デフォルトとしてリスト更新可オプションが無効になります。
DISPLAY SELECTION コマンドでは、この引数によりリストの値の選択が許可されるだけであり、変更は許可されないということを覚えておいてください。実際 DISPLAY SELECTION により、カレントテーブルは読み込みのみ状態になります。MODIFY SELECTIONを使用した場合にのみ、実際に値を入力することができます。

Note: OBJECT SET ENTERABLEコマンドを使用し、オンザフライでリスト更新可オプションを有効、または無効に設定できます。

オプションの * 引数に関する規則を次に説明します:
- セレクションにレコードが1件しか存在しないときに、1番目のオプションの * を指定しない場合、そのレコードは出力フォームではなく入力フォーム上に表示されます。
- 1番目のオプションの * を指定した場合は、セレクションに1件しかレコードが存在しない場合でも出力フォームを使用します
- 1番目のオプションの * を指定し、ユーザがレコードをダブルクリックしてそれを入力フォームに表示した場合、入力フォームのスクロールバーは表示されません。これを無効にするには、2番目のオプションの * を指定します。

DISPLAY SELECTION の実行を終了するため、出力フォームのフッタまたはヘッダエリアにカスタムボタンを配置できます。入力またはキャンセル自動アクション、あるいはACCEPTやCANCELコマンドを呼び出すオブジェクトメソッドを利用できます。DISPLAY SELECTION により呼び出された出力フォームにボタンが存在しない場合、**Escape** (Windows) または**Esc** (Mac OS) を使用してリストを終了できます。

DISPLAY SELECTION の実行中と後、ユーザが選択したレコードは UserSet という名前のセットに格納されます。UserSet は、セレクションの表示中にボタンがクリックされて呼び出されるオブジェクトメソッドや、メニュー項目が選択された際に実行されるメソッド内で使用できます。またコマンド終了後に、DISPLAY SELECTION を呼び出したプロジェクトメソッド内でも利用できます。

例題 1

以下の例は、最初に、[People]テーブルの全レコードをカレントセレクションにします。次にDISPLAY SELECTIONを使用してレコードを表示し、ユーザがプリントするレコードを選択します。最後に、選択されたレコードをUSE SETでカレントセレクションにし、PRINT SELECTIONコマンドでそのレコードを印刷します:

```
ALL RECORDS([People]) ` すべてのレコードを選択
DISPLAY SELECTION([People];*) ` レコード表示
USE SET("UserSet") ` ユーザが選択したレコードのみを使用
PRINT SELECTION([People]) ` ユーザが選択したレコードを印刷
```

例題 2

Form eventの6番目の例題を参照してください。この例題ではDISPLAY SELECTIONコマンドの実行中に発生するイベントをすべて監視するためのあらゆるチェックが示されています。

例題 3

アプリケーションモードでDISPLAY SELECTIONやMODIFY SELECTIONを使用して、デザインモードでの**レコード**メニューの機能を再現するには、以下の手順にしたがってください:

a. デザインモードで必要なメニューを備えたメニューバーを作成します。例えばすべて表示、クエリ、並び替え等のメニューです。

b. このメニューバーを (フォームプロパティダイアログボックスの連結するメニューバーでメニューを選択して)、DISPLAY SELECTIONやMODIFY SELECTIONで使用される出力フォームに関連付けます。

c. 以下のプロジェクトメソッドをメニューに関連付けます:

```
  ` M_SHOW_ALL (すべてを表示メニューに割り当て)
$vpCurTable:=Current form table
ALL RECORDS($vpCurTable->)
  ` M_QUERY (クエリメニューに割り当て)
$vpCurTable:=Current form table
QUERY($vpCurTable->)
  ` M_ORDER_BY (並び替えメニューに割り当て)
$vpCurTable:=Current form table
ORDER BY($vpCurTable->)
```

アプリケーションモードでセレクションの表示や修正を実行するたび、標準のメニューオプションを提供するために、PRINT SELECTIONやQR REPORT等、他のコマンドも使用できます。Current form table コマンドを使用すればこれらのメソッドは汎用コードとなり、このメニューバーをあらゆるテーブルのあらゆる出力フォームに関連付けることができます。

Displayed line number

Displayed line number -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	表示中の行番号

説明

Displayed line number コマンドはOn Display Detailフォームイベントでのみ機能します。このコマンドはレコードリストまたはリストボックスで画面に行が表示される際、処理中の行の番号を返します。Displayed line numberがリストまたはリストボックス表示以外の場面で呼び出されると、0を返します。

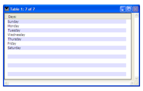
レコードリストの場合、表示された行が空でなければ (行がレコードに関連付けられている場合)、Displayed line numberから返される値はSelected record numberから返される値と同じです。

Selected record numberと同様、Displayed line numberは1から始まります。このコマンドは、空の行も含め、画面上に表示されたリストフォームやリストボックスの各行を処理したい場合に役立ちます。

例題

次の例題により、画面上に表示されるリストフォームに対し、レコードが表示されない行に対しても代替色を割り当てることができます:

```
  // リストフォームメソッド
If(Form event=On Display Detail)
   If(Displayed line number% 2=0)
  // 偶数行は白地に黒
      OBJECT SET RGB COLORS([Table1]Field1;-1;0x00FFFFFF)
   Else
  // 奇数行は明るい青地に黒
      OBJECT SET RGB COLORS([Table1]Field1;-1;0x00E0E0FF)
   End if
End if
```



Days:
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday

End selection

End selection {(aTable)} -> 戻り値

引数	型		説明
aTable	テーブル	→	レコードポインタがセレクションの最後のレコードよりも 後ろにあるかテストするテーブル、または 省略時、デフォルトテーブル
戻り値	ブール	←	Yes (TRUE) or No (FALSE)

説明

End selectionは、カレントレコードポインタがaTableのカレントセレクションの後ろにある場合TRUEを返します。一般にEnd selectionは、NEXT RECORDコマンドで、カレントレコードポインタが最後のレコードの後に移動したかどうかをチェックするために使用します。カレントセレクションが空の場合、End selectionはTRUEを返します。

カレントレコードのポインタをセレクション内に戻すには、LAST RECORD、FIRST RECORD、GOTO SELECTED RECORDを使用します。PREVIOUS RECORDでは、ポインタはセレクション内に戻りません。

PRINT SELECTIONまたはプリント...メニューを選択してレポートを印刷する場合、End selectionは最後のフッタでTRUEを返します。以下のステートメントを使用して、最後のフッタを判定して最終ページに特殊なフッタを印刷することができます:

```
  ` 集計レポートの出力用フォームのフォームメソッド
$vpFormTable:=Current form table
Case of
  ` ...
   :(Form event=On Printing Footer)
  ` フッタが印刷されようとしている
      If(End selection($vpFormTable->))
  ` 最後のフッタ用のコード
      Else
  ` フッタ用のコード
      End if
End case
```

例題

以下のフォームメソッドはレポートの印刷中に使用します。vFooter変数を設定し、最終ページのフッタエリアに印刷します:

```
  ` [Finances];"Summary" Form Method
Case of
  ` ...
   :(Form event=On Printing Footer)
      If(End selection([Finances]))
         vFooter:="c2001 Acme Corp."  ` 最後のページのフッタ
      Else
         vFooter:=""  ` 他のページのフッタ
      End if
End case
```

FIRST RECORD

FIRST RECORD {(aTable)}

引数	型		説明
aTable	テーブル	→	セレクションの先頭をカレントレコードにするテーブル または省略時デフォルトテーブル

説明

FIRST RECORDは、aTableのカレントセレクションの先頭レコードをディスクからロードし、カレントレコードに設定します。すべての検索、選択、ソートコマンドも先頭のレコードをカレントレコードに設定します。カレントセレクションが存在しない場合や、カレントレコードが既にセレクションの最初のレコードである場合、FIRST RECORDは何も行いません。

このコマンドはUSE SETコマンドの後で、レコードセレクションを最初のレコードからループする際によく使用されます。しかし、カレントレコードが実際に先頭のレコードであるかどうかが確かではない場合にも呼び出すことができます。

例題

以下の例は、[Customers]テーブルの最初のレコードをカレントレコードにします:

```
FIRST RECORD([Customers])
```

GET HIGHLIGHTED RECORDS

GET HIGHLIGHTED RECORDS ({aTable ;} setName)

引数	型		説明
aTable	テーブル	⇒	ハイライトされたレコードを読みだすテーブル 省略した場合、カレントフォームのテーブル
setName	文字	⇒	ハイライトしたレコードを格納するセット

説明

GET HIGHLIGHTED RECORDS コマンドは、aTable中で (例: リストフォームでユーザにより選択されて) ハイライトされたレコードをsetNameで指定したセットに格納します。aTableを省略すると、カレントフォームまたはサブフォームのテーブルが使用されます。

デザインモードまたはDISPLAY SELECTION / MODIFY SELECTIONコマンドを実行している時、このコマンドは4Dが自動で管理する**UserSet**を使用した呼び出しと置き換えることができます。しかしこのコマンドではハイライトさせたテーブルを選択することができるので、GET HIGHLIGHTED RECORDSを使用すればサブフォームのハイライトされたレコードを管理することも可能です。この場合、異なるテーブルのサブフォームのセレクションを扱うこともできます。**UserSet**に関する詳細はの節を参照してください。

GET HIGHLIGHTED RECORDSコマンドはフォーム以外のコンテキストでも呼び出せますが、その場合は空のセットが返されます。
setNameで指定するセットは、ローカル/クライアント、プロセス、またはインタープロセスのいずれでも構いません。

Note: 組み込んだサブフォームのプロパティで選択モードに**複数**を指定しない場合、GET HIGHLIGHTED RECORDSは空のセットを返します。この場合、選択されたレコードの行を知るにはSelected record numberコマンドを使用します。

例題

次のメソッドは、テーブル ([CDs]) のレコードを表示するサブフォームにおいて、選択されているレコードの数を示します:

```
GET HIGHLIGHTED RECORDS([CDs];"$highlight")
ALERT(String(Records in set("$highlight"))+" selected records.")
CLEAR SET("$highlight")
```

システム変数およびセット

コマンドが正しき実行されるとシステム変数OKに1が、そうでなければ0が設定されます。

GOTO SELECTED RECORD

GOTO SELECTED RECORD ({aTable ;} record)

引数	型		説明
aTable	テーブル	→	指定したレコードをカレントレコードとするテーブル, または 省略時、デフォルトテーブル
record	倍長整数	→	セレクション中のレコード位置番号

説明

GOTO SELECTED RECORDは、aTableのカレントセレクション内の指定されたレコードに移動し、そのレコードをカレントレコードにします。カレントセレクションは変更されません。recordにはカレントセレクション内のレコードの位置を指定します。Record numberで求められるレコード番号ではありません。このレコード位置はセレクションの作成方法およびセレクションがソートされているかどうかによって変わります。

GOTO SELECTED RECORD は以下の場合なにも行いません:

- カレントセレクション中にレコードが存在しない場合
- record がカレントセレクションに含まれていない場合
- record がすでにカレントレコードである場合

recordに0を渡すと、aTableのカレントレコードが存在しなくなります。単一選択モードが選択されている場合、特に組み込みサブフォームで、これによりリスト中のレコードの選択を解除できます。

例題

以下の例は、[People]Last Nameフィールドの内容をatNames配列に取り込みます。alRecNum倍長整数配列にレコード位置番号を設定します。両方の配列をソートし、その配列を使用してセレクション内のレコードを参照します:

```
  ` ここで[People] テーブルのセレクションを作成
  ` ...
  ` 名前を取得
SELECTION TO ARRAY([People]Last Name;atNames)
  ` レコード位置番号の配列を作成
$vlNbRecords:=Size of array(atNames)
ARRAY LONGINT(alRecNum;$vlNbRecords)
For($vlRecord;1;$vlNbRecords)
   alRecNum{$vlRecord}:=$vlRecord
End for
  ` 配列を名前順でソート
SORT ARRAY(atNames;alRecNum;>)
```

atNames 配列がスクロールエリアに表示され、ユーザは項目の一つをクリックします。2つの配列は同期されているので、alRecNumは対応するatNamesの要素のレコード位置番号を保持しています。

以下のatNamesオブジェクトメソッドは[People]セレクション中の、スクロールエリアで選択されたレコードをカレントレコードにします:

```
Case of
   :(Form event=On Clicked)
      If(atNames#0)
         GOTO SELECTED RECORD(alRecNum{atNames})
      End if
End case
```

HIGHLIGHT RECORDS

HIGHLIGHT RECORDS ({aTable }{;}{ setName {; *}})

引数	型	説明
aTable	テーブル	⇒ レコードをハイライトするテーブル 省略時、カレントフォームのテーブル
setName	文字	⇒ ハイライトさせるレコードのセット、または 省略時、Userset
*	演算子	⇒ リストの自動スクロールを無効

説明

HIGHLIGHT RECORDS コマンドは、出力フォーム内で指定されたレコードをハイライトします。この動作は、クリックまたは**Shift+クリック**、**Ctrl+クリック** (Windows) や**Command+クリック** (Mac OS) キーコンビネーションを使用し、リストモードでレコードを手動で選択する場合と同じです。カレントセレクションは変更されません。

Note: "選択"されたレコードのセットは、再描画後に更新されます。つまりメソッド全体が実行終了した後であり、HIGHLIGHT RECORDSを実行した直後ではありません。

aTable引数を使用し、"ハイライト表示"されるレコードが属するテーブルを指定します。この引数は、特にカレントテーブルに属していない組み込みサブフォームのレコードをハイライト表示するために使用することができます (後述参照)。

- setNameに有効なセット名を渡すと、コマンドは定義されたテーブルの、そのセット内のレコードに適用されます。
- setNameを省略すると、コマンドは現在のUserSetのレコードをハイライト表示します。このセットはユーザーモードとMODIFY SELECTION / DISPLAY SELECTIONを呼び出したときにのみ管理されます。サブフォームに対してレコードのハイライトを行いたい場合、テーブル名とセット名を指定する必要があります。UserSetセットについてはセットの節を参照してください。

* 引数を渡すと、ハイライト表示されたレコードが表示されていない場合に、そのリストの自動スクロール機能が無効になります。このメカニズムにより、OBJECT SET SCROLL POSITIONコマンドを使用して、独自のスクロール管理を行えるようになります。

Note: 組み込みサブフォームの場合、サブフォームの選択モードプロパティで**複数**が選択されていない場合、HIGHLIGHT RECORDS コマンドは何も行いません。この場合、行をハイライトするにはGOTO SELECTED RECORD コマンドを使用します。

例題

MODIFY SELECTIONコマンドによって表示される出力フォーム内で、カレントセレクションを変更することなく、ユーザが検索を実行できるようにしたいとします。これを実行するには、フォーム内に**検索** ボタンを置いて、押された時に下記のメソッドを実行します:

```
SET QUERY DESTINATION(Into set;"UserSet")
QUERY
SET QUERY DESTINATION(Into current selection)
HIGHLIGHT RECORDS
```

ユーザがボタンをクリックすると標準の検索ダイアログボックスが表示され、検索が実行された後、カレントセレクションを変更することなく、見つかったレコードを反転表示されます。

LAST RECORD

LAST RECORD {(aTable)}

引数	型	説明
aTable	テーブル	→ カレントレコードをセレクションの最後に移動する テーブル、または省略時、デフォルトテーブル

説明

LAST RECORDは、aTableのカレントセレクションの最後のレコードをディスクからロードし、カレントレコードに設定します。カレントセレクションが空の場合、または当該レコードが既にカレントレコードである場合、**LAST RECORD**は何も行いません。

例題

以下の例は、[People]テーブルの最後のレコードをカレントレコードにします。:

```
LAST RECORD([People])
```

MODIFY SELECTION

MODIFY SELECTION ({aTable}{; selectMode}{; enterList}{; *}{; *})

aTable	テーブル	⇒	表示および更新を行うテーブル, または 省略時、デフォルトテーブル
selectMode	倍長整数	⇒	選択モード
enterList	ブール	⇒	リスト入力を許可するオプション
*		⇒	1レコードセレクションの場合にも出力フォームを使用し、入力フォームのスクロールバーを隠す

説明

MODIFY SELECTIONは、DISPLAY SELECTIONとほぼ同じ機能を提供します。詳細についてはDISPLAY SELECTIONの説明を参照してください。2つのコマンドの違いを以下にあげます:

1. DISPLAY SELECTIONとMODIFY SELECTIONはカレントレコードセレクションをリストモードで表示し、ユーザがレコードをダブルクリックすると、そのレコードは入力フォーム上に表示されます。MODIFY SELECTIONを使用した場合は、別のプロセスやユーザ、またはリスト更新可モード (許可されている場合) でレコードが既に使用されていなければ、ダブルクリックしてそのレコードを修正することができます。
2. DISPLAY SELECTIONはカレントプロセスでレコードを読み込み専用でロードします。つまり、レコードは他のプロセスに対しロックされません。MODIFY SELECTIONはカレントプロセスでテーブルが読み書きに設定されているばあい、レコードを読み書き状態でロードします。つまり他のプロセスがレコードをロックしていなければ、そのレコードはこのプロセス用に他のプロセスに対しロックされます。MODIFY SELECTION実行が終了後、レコードは解放されます。

NEXT RECORD

NEXT RECORD {(aTable)}

引数	型		説明
aTable	テーブル	→	カレントレコードをセレクションの次のレコードに 移動するテーブル、省略時はデフォルトテーブル

説明

NEXT RECORDは、カレントプロセスのaTableのカレンセレクションにある次のレコードへカレントレコードポインタを移動します。カレントセレクションが空の場合、あるいはBefore SelectionまたはEnd selectionがTRUEの場合、NEXT RECORDは何も行いません。

NEXT RECORDでカレントセレクションの最後を超えてカレントレコードポインタを移動した場合、End selectionはTRUEを返しカレントレコードはなくなります。この場合、FIRST RECORD、LAST RECORD、GOTO SELECTED RECORDコマンドを使用して、カレントレコードポインタをカレントセレクション内に戻します。

例題

DISPLAY RECORDの例題参照

ONE RECORD SELECT

ONE RECORD SELECT {(aTable)}

引数	型		説明
aTable	テーブル	→	カレントレコードをカレントセレクションにする テーブル、または省略時デフォルトテーブル

説明

ONE RECORD SELECTはaTableのカレントレコードをカレントセレクションにします。カレントレコードが存在しないかカレントレコードがメモリにロードされていない場合 (特殊なケース)、ONE RECORD SELECTは何も行いません。

注

このコマンドはレコードスタックにプッシュしてポップしたレコードを、セレクションが変更された際にカレントセレクションにするために使用されました。バージョン6で、SET QUERY DESTINATIONを使用してセレクションやカレントレコードを変更せずに検索を行えるようになりました。これにより同じテーブルをクエリする目的でレコードをプッシュ/ポップする必要はなくなりました。結果は、セレクションをカレントレコードだけにしたい場合を除いて利用価値が少なくなりました。

PREVIOUS RECORD

PREVIOUS RECORD {(aTable)}

引数	型		説明
aTable	テーブル	→	セレクションの前レコードをカレントレコードにする テーブル、省略時はデフォルトテーブル

説明

PREVIOUS RECORDは、カレントプロセスのaTableのカレントセレクションにある1つ前のレコードへカレントレコードポインタを移動します。カレントセレクションが空の場合、またはBefore selectionやEnd selectionがTRUEの場合、PREVIOUS RECORDは何も行いません。

PREVIOUS RECORDで、カレントセレクションの前にカレントレコードポインタを移動した場合は、Before selectionはTRUEを返し、カレントレコードはなくなります。この場合、FIRST RECORD、LAST RECORD、GOTO SELECTED RECORDを使用して、カレントレコードポインタをカレントセレクション内に戻します。

Records in selection

Records in selection {(aTable)} -> 戻り値

引数	型		説明
aTable	テーブル	→	カレントセレクション数を返すテーブル 省略時、デフォルトテーブル
戻り値	倍長整数	⮐	カレントセレクションのレコード数

説明

Records in selectionは、aTableのカレントセレクションのレコード数を返します。一方、Records in tableはテーブルの全レコード数を返します。

例題

以下の例は、カレントセレクションのすべてのレコード間を移動するのによく使用されるループ処理です。同じ動作はAPPLY TO SELECTIONコマンドで行うこともできます:

```
FIRST RECORD([People]) ` セレクションの最初のレコードから開始
For($vlRecord;1;Records in selection([People])) ` レコード毎にループ
   Do Something ` レコードに処理を行う
   NEXT RECORD([People]) ` 次のレコードに移動
End for
```

REDUCE SELECTION

REDUCE SELECTION ({aTable ;} number)

引数	型		説明
aTable	テーブル	→	セレクションの数を減らすテーブル, または 省略時デフォルトテーブル
number	倍長整数	→	選択するレコード数

説明

REDUCE SELECTIONは、aTableの新しいレコードセレクションを作成します。このコマンドはaTableのカレントセレクションの先頭からnumber個のレコードセレクションを返します。REDUCE SELECTIONは、カレントプロセスのaTableのカレントセレクションに適用されます。これはカレントプロセスのaTableのカレントセレクションを変更し、新しいセレクションの先頭レコードをカレントレコードにします。

Note: ステートメントREDUCE SELECTION(0)が実行されると、aTableのカレントセレクションおよびカレントレコードはなくなります。

例題

以下の例では最初に、20を超える国の販売店を対象にしたコンテストの正確な統計を検索します。国ごとに$50,000以上の製品売上を記録した上位3店と、全世界で上位100店に含まれる販売店に対し、賞が送られます。ほんの数行のコードで、この複雑な処理がインデックス検索を利用して実行されます:

```
CREATE EMPTY SET([Dealers];"Winners") ` 空のセットを作成
SCAN INDEX([Dealers]Sales amount;100;<) ` インデックスの最後からスキャン
CREATE SET([Dealers];"100 best Dealers") ` 選択されたレコードをセットに格納
For($Country;1;Records in table([Countries])) ` 国ごとに
  ` この国の販売店を検索
    QUERY([Dealers];[Dealers]Country=[Countries]Name;*) ` ...$50,000以上売った販売店
    QUERY(&;[Dealers];[Dealers]Sales amount>=50000)
    CREATE SET([Dealers];"WinnerDealers") ` セットに格納
  ` 上位100位以内のセットに結合
    INTERSECTION("WinnerDealers";"100 best Dealers";"WinnerDealers")
    USE SET("WinnerDealers") ` 国ごとの上位成績者
  ` 降順で並び替え
    ORDER BY([Dealers];[Dealers]Sales amount;<)
    REDUCE SELECTION([Dealers];3) ` 3位までを選択
    CREATE SET([Dealers];"WinnerDealers") ` その国の勝者
  ` 全世界の勝者リストに加える
    UNION("WinnerDealers";"TheWinners";"TheWinners")
End for
CLEAR SET("100 best Dealers") ` このセットはもう必要ない
CLEAR SET("WinnerDealers") ` このセットも必要ない
USE SET("The Winners") ` これが勝者
CLEAR SET("The Winners") ` このセットはもう必要ない
OUTPUT FORM([Dealers];"Prize letter") ` 印刷フォームを選択
PRINT SELECTION([Dealers]) ` 手紙を印刷
```

SCAN INDEX

SCAN INDEX (aField ; number {; > または <})

引数	型		説明
aField	フィールド	⇒	インデックスをスキャンするインデックスフィールド
number	倍長整数	⇒	返すレコード数
> または <	演算子	⇒	>: インデックスの始まりから <: インデックスの終わりから

説明

SCAN INDEXは、aField フィールドを含むテーブルからnumber個のレコードのセレクションを作成します。<を渡した場合、SCAN INDEXはインデックスの最後からnumber個のセレクション作成します。>を渡した場合、SCAN INDEXはインデックスの先頭からnumber個のセレクションを作成します。このコマンドは、インデックスを用いるため非常に効率が良くなります。

Note: 結果のセレクションは、ソートされていません。

SCAN INDEXは、インデックスフィールドにのみ使用できます。このコマンドはカレントプロセスのテーブルのカレントセレクションを変更し、セレクションの先頭レコードをカレントレコードとしてロードします。

テーブル内のレコード数より多くのレコードを指定した場合、SCAN INDEXはすべてのレコードを含むセレクションを作成します。

例題

以下の例は、ワースト50の顧客とベスト50の顧客に手紙を出します:

```
SCAN INDEX([Customers]TotalDue;50;<) ` ワースト50の顧客を得る
ORDER BY([Customers]Zipcode;>) ` 郵便番号で並び替え
FORM SET OUTPUT([Customers];"ThreateningMail")
PRINT SELECTION([Customers]) ` 手紙を印刷
SCAN INDEX([Customers]TotalDue;50;>) ` ベスト50の顧客を得る
ORDER BY([Customers]Zipcode;>) ` 郵便番号で並び替え
FORM SET OUTPUT([Customers];"Thanks Letter")
PRINT SELECTION([Customers]) ` 手紙を印刷
```

Selected record number

Selected record number {(aTable)} -> 戻り値

引数	型		説明
aTable	テーブル	→	レコード位置番号を取得するテーブル、または 省略時はデフォルトテーブル
戻り値	倍長整数	↩	カレントレコードのレコード位置番号

説明

Selected record numberは、aTableのカレントセレクション内でのカレントレコードの位置を返します。

セレクションが空ではなく、カレントレコードがそのセレクションに含まれるときに、Selected record numberは1からRecords in selectionまでの値を返します。セレクションが空かカレントレコードが存在しない場合、この関数は0を返します。

レコード位置番号は、Record numberで求めるレコード番号とは異なります。Record numberは絶対レコード番号を返します。レコード位置番号は、カレントセレクションおよびカレントレコードに依存します。

例題

以下の例は、カレントレコードのレコード位置番号を変数に格納します:

```
CurSelRecNum:=Selected record number([People])  ` レコード位置番号を取得
```

TRUNCATE TABLE

TRUNCATE TABLE {(aTable)}

引数	型	説明
aTable	テーブル	→ すべてのレコードが削除されるテーブル 省略時はデフォルトテーブル

説明

TRUNCATE TABLE コマンドはaTableのすべてのレコードを素早く削除します。aTableが既に空の場合、TRUNCATE TABLEは何も行いません。コマンドの呼出し後、カレントセレクションやカレントレコードはありません。

このコマンドの効果はALL RECORDSとDELETE SELECTIONの呼び出しと同じです。しかしその動作は以下の点で異なります:

- トリガは呼び出されません。
- データの参照整合性はチェックされません。
- TRUNCATE TABLEを実行するプロセスはトランザクション中であってはなりません。トランザクション中の場合、コマンドは何も行わず、OKシステム変数に0が設定されます。
- 1つ以上のレコードが他のプロセスによりロックされていると、コマンドは失敗します。エラーが生成され、OKシステム変数に0が設定されます。**LockedSet**システムセットは作成されません。
- aTableが既に空の場合、TRUNCATE TABLEは何も行わず、OK変数は1に設定されます。
- aTableが読み込みのみの場合、TRUNCATE TABLEは何も行わず、OK変数は0に設定されます。
- ログファイルがあれば、処理はログファイルに記録されます。

TRUNCATE TABLEコマンドは注意して使用しなければなりませんが、例えば一時的なデータを素早く削除するなど特定のケースではとても効率的です。

Note: このコマンドのコンセプトと動作はSQL TRUNCATE (テーブル) コマンドと同じです。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKは1に、そうでなければ0に設定されます。

ツール

- BASE64 DECODE
- BASE64 ENCODE
- Choose
- Generate digest Updated 14.0
- Generate UUID
- GET ACTIVITY SNAPSHOT New 14.0
- GET MACRO PARAMETER
- LAUNCH EXTERNAL PROCESS
- OPEN WEB URL
- PROCESS 4D TAGS
- SET ENVIRONMENT VARIABLE
- SET MACRO PARAMETER

BASE64 DECODE

BASE64 DECODE ({encodedText ;} BLOB)

引数	型		説明
encodedText	テキスト	→	Base64フォーマットにエンコードされたBLOBを含むテキスト
BLOB	BLOB	→	Base 64フォーマットでコード化されているBLOB
		←	解読されたBLOB

説明

BASE64 DECODEコマンドを使用して、encodedTextまたはblob引数に渡された、Base64フォーマットにコード化されたBLOBをデコードできます。

encodedText引数を渡すと、コマンドはその内容をデコードしてblob引数に返します。encodedText引数にはBase64フォーーマットにエンコードされたBLOBが含まれていなければなりません。この場合、blob引数の最初の内容は無視されます。
encodedText引数を省略すると、コマンドは直接blob引数に渡されたBLOBを更新します。

コマンドはencodedTextやblob引数の内容を検証しません。渡されたデータが実際にBase64フォーマットでコード化されているかどうかは開発者が検証しなければなりません。そうでなければ結果は正しいものではないかもしれません。

例題

この例題ではBLOBを使用してピクチャーを転送します:

```
C_BLOB($sourceBlob)
C_PICTURE($mypicture)
$mypicture:=[people]photo
PICTURE TO BLOB($mypicture;$sourceBlob;".JPG")
C_TEXT($base64Text)
BASE64 ENCODE($sourceBlob;$base64Text) //テキストにエンコード
  // バイナリは文字列として$base64Textに格納されている

C_TEXT($base64Text)
C_BLOB($targetBlob)
BASE64 DECODE($base64Text;$targetBlob) //テキストをデコード
  // Base64にエンコードされたバイナリは$blobTargetにデコードされる
```

BASE64 ENCODE

BASE64 ENCODE (BLOB {; encodedText})

引数	型		説明
BLOB	BLOB	→	Base 64フォーマットでコード化するBLOB
		←	Base 64フォーマットでコード化したBLOB
encodedText	テキスト	←	Base64フォーマットにエンコードされたBLOBのテキスト

説明

BASE64 ENCODEコマンドはblob引数に渡されたBLOBをBase64フォーマットにエンコードします。

encodedText引数を渡すと、コマンド実行後、エンコードされたblobの内容をテキストとして受け取ります。この場合、blob自身は変更されません。encodedText引数を省略すると、コマンドは直接引数として渡されたBLOBを更新します。

Base64エンコードは8ビットデータを7ビットにコード化します。このエンコーディングは例えばXMLでBLOBを扱う際に必要となります。

Choose

Choose (criterion ; value {; value2 ; ... ; valueN}) -> 戻り値

引数	型		説明
criterion	ブール, 整数	→	テストする値
value	式	→	可能な値
戻り値	式	↩	条件の値

説明

Chooseコマンドは、引数criterionの値に応じて、引数value1、value2などに渡された値の1つを返します。

ブールまたは数値タイプのいずれかを引数criterionに渡します。

- criterionがブールの場合、ChooseはブールがTrueのときvalue1を、Falseのときvalue2を返します。この際コマンドは3つの引数、criterion、value1とvalue2を期待します。
- criterionが整数の場合、Choose はcriterionに対応する位置にある値を返します。値の採番は0から始まりますので注意してください (従ってvalue1の位置は0です) 。この場合、コマンドは少なくとも2つの引数、criterionとvalue1を期待します。

コマンドは、引数value にすべてのデータタイプを受け入れます。しかしピクチャ、ポインタ、BLOBSおよび配列は除外です。それでも、渡されたすべての値が同じタイプであることを確かめる必要があります。この点において、4Dは照合を実行しません。

criterionに対応する値がない場合、Chooseは引数valueのタイプに対応する空値を返します (例えば数値タイプは0、文字列タイプは""など) 。

このコマンドを使用して簡単なコードを作成し、複数の行から成るCase ofタイプのテストを置き換えることができます (例2を参照) 。これはフォーミュラが実行される場所でも非常に役に立ちます。クエリエディタ、フォーミュラのアプリケーション、クイックレポートエディタ、リストボックスで計算されるカラムなどです。

例題 1

ブール型条件を用いた、このコマンドの典型的な使用例を以下に示します。

```
vTitle:=Choose([Person]Masculine;"Mr";"Ms")
```

このコードは以下のコードと完全に対等です。

```
If([Person]Masculine)
   vTitle:="Mr"
Else
   vTitle:="Ms"
End if
```

例題 2

数値型条件を用いた、このコマンドの典型的な使用例を以下に示します。

```
vStatus:=([Person]Status;"Single";"Married";"Widowed";"Divorced")
```

このコードは以下のコードと完全に対等です。

```
Case of
   :([Person]Status=0)
      vStatus:="Single"
   :([Person]Status=1)
      vStatus:="Married"
   :([Person]Status=2)
      vStatus:="Widowed"
   :([Person]Status=3)
      vStatus:="Divorced"
End case
```

Generate digest

Generate digest (param ; algorithm) -> 戻り値

引数	型		説明
param	BLOB, テキスト変数	→	Digestキーを取得するBLOBやテキスト
algorithm	倍長整数	→	キーの生成に使用するアルゴリズム 0= MD5、1 = SHA1、2=4Dダイジェスト
戻り値	テキスト	↩	Digestキーの値

説明

Generate digestコマンドはBLOBやテキストのDigestキーを指定したアルゴリズムで生成して返します。

現在4Dでは2つのアルゴリズム、**MD5** (*Message Digest 5*) と **SHA-1** (*Secure Hash 1*) と **4D** (内部アルゴリズム)が利用できます。これらのアルゴリズムは異なるハッシュ関数です:

- MD5では16 byteの値が計算され、16進形式で32文字が返されます。
- SHA-1では20 byteの値が計算され、16進形式で40文字が返されます。
- 4D では4D内部でユーザーパスワードを暗号化するために使用される内部アルゴリズムを使用します。このアルゴリズムは MissingRef のコンテキストで独自のユーザーリストを使用したいときに特に有効です。

同じオブジェクトに対してはすべてのプラットフォーム (Mac/Windows, 32/64 bits) で同じ値が返されます。 計算は引数に渡されたテキストのUTF-8での表記に基づいて実行されます。

注: コマンドを空のテキストやBLOBに対して実行すると、計算結果として以下が返されます (エラーにはなりません):
"d41d8cd98f00b204e9800998ecf8427e" (MD5)
"da39a3ee5e6b4b0d3255bfef95601890afd80709" (SHA-1)

param 引数にはテキストまたはBLOBフィールドや変数を渡します。**Generate digest**コマンドはダイジェストキーを文字列として返します。

algorithm 引数で使用するハッシュ関数を指定します。Digest Typeテーマの以下の定数を使用できます:

定数	型	値	コメント
4D digest	倍長整数	2	4D内部のアルゴリズムを使用
MD5 Digest	倍長整数	0	MD5アルゴリズムを使用
SHA1 Digest	倍長整数	1	SHA-1アルゴリズムを使用

ダイジェストキーの計算が失敗した場合コマンドはエラーを生成し、空の文字列を返します。このエラーはON ERR CALLでインストールされるエラー処理メソッドで処理できます。

例題 1

この例題ではMD5アルゴリズムを使用して2つのドキュメントを比較します:

```
PLATFORM PROPERTIES($Platf;$Syst;$vlMachine)
 // 一番目のドキュメントを読み込みモードで開く
$Same:=True
$vhDocRef1:=Open document("";"*";Read Mode))
If(OK=1) // ドキュメントが選択されたら
   DOCUMENT TO BLOB(Document;$FirstBlob) // ドキュメントをロード
   If(OK=1)
      If($Platf=Mac OS)
         DOCUMENT TO BLOB(Document;$FirstBlobRF;*)
 // Mac OSではリソースフォークもロード
         $MD5_1RF:=Generate digest($FirstBlobRF;MD5 Digest)
      End if

 // 二番目のドキュメントを読み込みモードで開く
      $vhDocRef2:=Open document("";"*";Read Mode))
      If(OK=1)
         DOCUMENT TO BLOB(Document;$SecondBlob)
         If(OK=1)
            If($Platf=Mac OS)
               DOCUMENT TO BLOB(Document;$SecondBlobRF;*)
               $MD5_2RF:=Generate digest($SecondBlobRF;MD5 Digest)
               If($MD5_1RF#$MD5_2RF) // ダイジェストを比較
                  $Same:=False
               End if
            End if
            $MD5_1:=Generate digest($FirstBlob;MD5 Digest)
            $MD5_2:=Generate digest($SecondBlob;MD5 Digest)
            If(($MD5_1#$MD5_2)|($Same=False))
               ALERT("異なるドキュメントです")
            End if
         End if
      End if
   End if
End if
```

例題 2

この例題ではテキストのダイジェストキーを取得します:

```
$key1:=Generate digest("The quick brown fox jumps over the lazy dog.";MD5 Digest)
 // $key1は"e4d909c290d0fb1ca068ffaddf22cbd0"
$key2:=Generate digest("The quick brown fox jumps over the lazy dog.";SHA1 Digest)
 // $key2は"408d94384216f890ff7a0c3528e8bed1e0b01621"
```

例題 3

この例題ではパスワード "123"を使用する、4Dユーザーと合致しない "admin"というユーザーのみを受け入れる場合を考えます:

```
 //On REST Authentication database method
C_TEXT($1;$2)
C_BOOLEAN($0;$3)
 //$1: ユーザー
 //$2: パスワード
 //$3: ダイジェストモード
If($1="admin")
   If($3)
      $0:=($2=Generate digest("123";4D digest))
   Else
      $0:=($2="123")
   End if
Else
   $0:=False
End if
```

Generate UUID

Generate UUID -> 戻り値

引数	型		説明
戻り値	文字	↩	新しい UUIDテキスト (非整形32 文字)

説明

Generate UUID は32文字のUUID識別子を整形されていない形式で返します。

UUIDは16バイト (128 bit) の数値で、32個の16進文字を含んでいます。UUIDは非整形 ([A-F, a-f]および[0-9]からなる32文字 (例: 550e8400e29b41d4a716446655440000)) または整形 (8,4,4,4,12のグループ (例: 550e8400-e29b-41d4-a716-446655440000))で表すことができます。

4DではUUID番号をフィールドに格納することができます。詳細はDesign ReferenceマニュアルのUUIDフォーマットを参照してください。

例題

UUIDを生成して変数に代入する:

```
C_TEXT(MyUUID)
MyUUID:=Generate UUID
```

GET ACTIVITY SNAPSHOT

GET ACTIVITY SNAPSHOT ({* ;} arrActivities | arrUUID ; arrStart ; arrDuration ; arrInfo {; arrSubOp})

引数	型		説明
*	演算子	⇒	渡した場合、サーバーの状態を取得
arrActivities | arrUUID	Object array, テキスト配列	⇐	オペレーションの詳細な情報(オブジェクト配列) またはオペレーションの UUID(テキスト配列)
arrStart	テキスト配列	⇐	オペレーションの開始時刻
arrDuration	倍長整数配列	⇐	オペレーションの所要時間(ミリ秒単位)
arrInfo	テキスト配列	⇐	オペレーションを説明するラベル
arrSubOp	Object array	⇐	subOperations プロパティ

説明

GET ACTIVITY SNAPSHOTコマンドは、4D data上で進行中の操作の詳細を記載した配列を、一つまたは複数の配列に返します。これらの操作は、通常進捗ウィンドウが表示されます。

この情報は、もっとも時間のかかっているオペレーションまたは頻繁に使用されているオペレーション(ライトキャッシュやフォーミュラの実行など)のスナップショットを取るのに使われます。

注: GET ACTIVITY SNAPSHOT コマンドによって返された情報は、4D Server のリアルタイムモニターのページに表示されているデータと同じです。(*4D Server Reference Guide* を参照して下さい。)

デフォルトとして、**GET ACTIVITY SNAPSHOT** はローカルに実行されている操作のみ処理します(4D シングルユーザー、4D サーバーまたはリモートモードの4Dなど)。それに加え、リモートモードの4Dでは、サーバーで実行されている操作のスナップショットを取ることもできます。そのためには第一引数として、* 演算子を渡して下さい。サーバーのデータはローカルに復元されます。
* 演算子は、コマンドが4Dサーバーや4Dシングルユーザー上で実行された場合には無視されます。

GET ACTIVITY SNAPSHOT コマンドではシンタックスを使用することができます:

- オブジェクト配列のみを使用するシンタックス
- 複数の配列を使用するシンタックス

第一シンタックス: GET ACTIVITY SNAPSHOT ({* ;} arrActivities)

この記法では、リアルタイムモニターの全てのオペレーションが構造化された形式で4D オブジェクト配列(arrActivities 引数で指定)に返されます。配列の各要素は、以下の様に構築されたオブジェクトになっています:

```
progressIndicator :{
    ID: string
    StartTime: 12:15:20:0054
    Duration: 501
    Title: String,
    sub-operations: [
        {
            Title: string,
            Start time: string,
            Duration: number (milliseconds)
        },
        {
            Title: string,
            Start time: string,
            Duration: number (milliseconds)
        }
    ]
}
```

第二シンタックス: GET ACTIVITY SNAPSHOT ({* ;} arrUUID ; arrStart ; arrDuration ; arrInfo {;arrSubOp})

この記法では、リアルタイムモニターの全てのオペレーションが、同期した複数の配列に返されます(オペレーションがあるたび、全ての配列に要素が追加されていきます)。返される配列は以下の通りです:

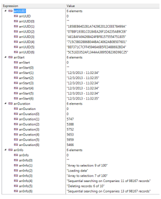
- arrUUID: それぞれのオペレーションに対する UUID が保存されます。
- arrStart: それぞれのオペレーションの開始時刻が、"dd/mm/yyyy - hh:mm:ss" というフォーマットで格納されます。(テキスト)
- arrDuration: それぞれのオペレーションの所要時間がミリ秒単位で格納されます。
- arrInfo : それぞれのオペレーションを説明するラベルが保存されます。
- arrSubOp (任意): この配列の要素には、"subOperations" プロパティを格納するオブジェクトが入っています。このプロパティの値はカレントオペレーションのサブ処理を全て含んだ object 配列となっています。カレントのオペレーションにサブ処理が何もない場合、subOperations の値は空の配列となります。

例題

4D か 4Dサーバーにおける個別のプロセスにおいて以下のメソッドを実行した場合、下図の様にオペレーションのスナップショットを返します:

```
ARRAY TEXT(arrUUID;0)
ARRAY TEXT(arrStart;0)
ARRAY LONGINT(arrDuration;0)
ARRAY TEXT(arrInfo;0)

Repeat
   GET ACTIVITY SNAPSHOT(arrUUID;arrStart;arrDuration;arrInfo)
   If(Size of array(arrUUID)>0)
      TRACE // デバッガを呼び出し
   End if
Until(False) // 無限ループ
```

以下の様な配列が返されます:



Expression	Value
arrUUID	6 elements
arrUUID	0
arrUUID{0}	""
arrUUID{1}	"1858EB64D281A7429E2012CEEE78499A"
arrUUID{2}	"07BBF193B1C0184EA26F1D4235A89CE6"
arrUUID{3}	"AE18AFA9426B424FBFB1575554751E05"
arrUUID{4}	"715C5B028868E44BAC4062AB0B507601"
arrUUID{5}	"8EF371C7CFF45946A685FE34886928D4"
arrUUID{6}	"5C51ED352AF1344AA3895D8236D9EC25"
arrStart	6 elements
arrStart	0
arrStart{0}	""
arrStart{1}	"12/3/2013 - 11:32:34"
arrStart{2}	"12/3/2013 - 11:32:35"
arrStart{3}	"12/3/2013 - 11:32:34"
arrStart{4}	"12/3/2013 - 11:32:34"
arrStart{5}	"12/3/2013 - 11:32:34"
arrStart{6}	"12/3/2013 - 11:32:35"
arrDuration	6 elements
arrDuration	0
arrDuration{0}	0
arrDuration{1}	5747
arrDuration{2}	5388
arrDuration{3}	5752
arrDuration{4}	5653
arrDuration{5}	5959
arrDuration{6}	5466
arrInfo	6 elements
arrInfo	0
arrInfo{0}	""
arrInfo{1}	"Array to selection: 9 of 100"
arrInfo{2}	"Loading data"
arrInfo{3}	"Array to selection: 7 of 100"
arrInfo{4}	"Sequential searching on Companies: 11 of 98167 records"
arrInfo{5}	"Deleting records: 6 of 10"
arrInfo{6}	"Sequential searching on Companies: 13 of 98167 records"

GET MACRO PARAMETER

GET MACRO PARAMETER (selector ; textParam)

引数	型		説明
selector	倍長整数	→	使用するセレクション
textParam	テキスト	←	返されたテキスト

説明

GET MACRO PARAMETERコマンドは、呼び出されたメソッドテキストのすべてまたは一部を引数textParamに返します。

引数selector を使用して、返される情報のタイプを設定します。テーマ"" に追加されている以下の定数の一つを渡します。

定数	型	値
Full method text	倍長整数	1
Highlighted method text	倍長整数	2

Full method textをselectorに渡すと、メソッドのテキストはすべてtextParamに返されます。Highlighted method textをselectorに渡すと、メソッドで選択されたテキストだけがtextParamに返されます。

例題

SET MACRO PARAMETERコマンドの例を参照してください。

LAUNCH EXTERNAL PROCESS

LAUNCH EXTERNAL PROCESS (fileName {; inputStream {; outputStream {; errorStream}}})

引数	型		説明
fileName	文字	→	ファイルパスと起動するファイルの引数
inputStream	文字, BLOB	→	入力ストリーム(stdin)
outputStream	文字, BLOB	←	出力ストリーム(stdout)
errorStream	文字, BLOB	←	エラーストリーム(stderr)

説明

LAUNCH EXTERNAL PROCESSコマンドを使用して、Mac OS XとWindowsで4Dから外部プロセスを起動させることができます。
Mac OS Xでは、コマンドを用いてターミナルから起動できる実行可能なアプリケーションへアクセスできます。

注: このコマンドは、4D Packの**AP_Sublaunch**コマンドと同じ機能 (と拡張された特徴) を持ちます。

実行するアプリケーションの固定されたファイルパスと (必要に応じて) 引数を引数fileNameに渡します。
Mac OS Xではアプリケーション名を渡すこともできます。4Dは環境変数**PATH**を使用して、実行可能ファイルを探します。

警告: このコマンドは実行可能なアプリケーションのみを起動することができます。シェル (コマンドインタプリタ) の一部である命令は実行できません。例えば、Mac OSではこのコマンドを用いてecho命令やインダイレクトを実行することはできません。

オプションのinputStream引数は外部プロセスのstdinを格納します。コマンドが実行されると、引数outputStreamとerrorStream (渡した場合) は外部プロセスのstdoutとstderrをそれぞれ返します。32KBまたはバイナリデータ(ピクチャのような) より大きなサイズのテキストデータを処理する場合、文字列の代わりにBLOB引数を使用します

注: SET ENVIRONMENT VARIABLEコマンド経由(非同期実行) で環境変数_4D_OPTION_BLOCKING_EXTERNAL_PROCESS を使用した場合、引数outputStreamとerrorStreamは返されません。

Mac OS Xでの例題

Application/Utilitiesフォルダにある Mac OS Xターミナルを使用します。

1. ファイルに対してパーミッションを変更する(chmod はファイルアクセスを変更するために使用するMac OS Xコマンドです)

```
LAUNCH EXTERNAL PROCESS("chmod +x /folder/myfile.txt")
```

2. テキストファイルを編集する(catはファイルを編集するために使用するMac OS X マンドです)。この例ではコマンドの完全なアクセスパスが渡されています。

```
C_TEXT(input;output)
input:=""
LAUNCH EXTERNAL PROCESS("/bin/cat /folder/myfile.txt";input;output)
```

3. "Users" フォルダの内容を取得する(ls -lはDOSのdir コマンド に相当するMac OS Xコマンドです)

```
C_TEXT($In;$Out)
LAUNCH EXTERNAL PROCESS("/bin/ls -l /Users";$In;$Out)
```

4. 独立している"グラフィック" アプリケーションを起動させるには、openシステムコマンドを使用するのが望ましいです(この場合、LAUNCH EXTERNAL PROCESS ステートメントはアプリケーションをダブルクリックすることと同じ効果があります) 。

```
LAUNCH EXTERNAL PROCESS("open /Applications/Calculator.app")
```

Windowsでの例題

5. NotePadを開く

```
LAUNCH EXTERNAL PROCESS("C:\\WINDOWS\\notepad.exe")
```

6. Notepadを開き、特定のドキュメントを開く

```
LAUNCH EXTERNAL PROCESS("C:\\WINDOWS\\notepad.exe C:\\Docs\\new folder\\res.txt")
```

7. MicrosoftR WordRアプリケーションを起動させて、特定のドキュメントを開く(2つの""を使用)

```
 $mydoc:="C:\\Program Files\\Microsoft Office\\Office10\\WINWORD.EXE \"C:\\Documents and
Settings\\Mark\\Desktop\\MyDocs\\New folder\\test.xml\""
 LAUNCH EXTERNAL PROCESS($mydoc;$tIn;$tOut)
```

8. Perlスクリプトを実行する(ActivePerlを必要とします):

```
C_TEXT($input;$output)
SET ENVIRONMENT VARIABLE("myvariable";"value")
LAUNCH EXTERNAL PROCESS("D:\\Perl\\bin\\perl.exe D:\\Perl\\eg\\cgi\\env.pl";$input;$output)
```

9. コンソールを表示せずにカレントディレクトリでコマンドを実行させる

```
SET ENVIRONMENT VARIABLE("_4D_OPTION_CURRENT_DIRECTORY";"C:\\4D_VCS")
SET ENVIRONMENT VARIABLE("_4D_OPTION_HIDE_CONSOLE";"true")
LAUNCH EXTERNAL PROCESS("mycommand")
```

10. ユーザが選択した外部ドキュメントをWindowsで開く

```
$docname:=Select document("";"*.*";"Choose the file to open";0)
If(OK=1)
   SET ENVIRONMENT VARIABLE("_4D_OPTION_HIDE_CONSOLE";"true")
   LAUNCH EXTERNAL PROCESS("cmd.exe /C start \"\" \""+$docname+"\"")
End if
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKは1に設定されます。そうでなければ (ファイルが見つからない、メモリが足りないなど) 0が設定されます。

OPEN WEB URL

OPEN WEB URL (url {; *})

引数型			説明
url	文字	⇒	開くURL
*	演算子	⇒	指定した場合 = URLをエンコードしない, 省略した場合 = URLをエンコードする

説明

OPEN WEB URLコマンドは、適切なアプリケーションを使用して、url 引数に渡したファイルやURLを開きます。

url 引数には標準のURLやファイルパス名を渡せます。

4Dはまず引数をファイルパス名として解釈しようとします。コマンドはMac OSでコロン (':') を、Windowsでバックスラッシュ ('\') を、またはfile://から始まるPOSIX URLを受け入れます。この場合4Dはシステムに、もっとも適切なアプリケーションを使用してファイルを開くよう、リクエストします (例えば、.htmlファイルにはブラウザを、.docファイルにはMS Wordを使用します)。この場合 * 引数は無視されます。

url 引数に標準のURL (mailto:, news:, http:などのプロトコル) が渡された場合、4D はデフォルトのWebブラウザを開始し、URLにアクセスします。コンピュータに接続されたボリュームにブラウザがない場合、このコマンドは何も行いません

4Dは自動でURLの特別文字をエンコードします。引数に*を渡すと、4DはURL特別文字のエンコードを行いません。このオプションを使用して、以下のようなURLの送信が可能です: "*http://www.server.net/page.htm?q=something*"

注: このコマンドはWebプロセスから呼ばれた時は動作しません。

例題 1

以下では、このコマンドがURL引数として受け入れる異なるタイプの文字列を例示します:

```
OPEN WEB URL("http://www.4d.com")
OPEN WEB URL("file://C:/Users/Lauren/Documents/pending.htm")
OPEN WEB URL("C:\Users\Lauren\Documents\pending.htm")
OPEN WEB URL("mailto:john_martin@4d.com")
```

例題 2

この例はアプリケーションを起動するために使用できます:

```
$fichier:=Select document("";"";0)
If(OK=1)
   OPEN WEB URL(Document)
End if
```

PROCESS 4D TAGS

PROCESS 4D TAGS (inputData ; outputData)

引数	型		説明
inputData	テキスト, BLOB	→	処理する4Dタグを格納しているデータ
outputData	テキスト, BLOB	←	処理されたデータ

説明

PROCESS 4D TAGSコマンドを使用すると、引数 inputData (BLOBまたはテキスト型の変数やフィールド) に格納されている4Dタグが処理され、その結果がoutputDataに返されます。

このコマンドにより、WebサーバーがWEB SEND BLOBタイプのコマンドや".shtml"拡張子付きのURLでリクエストされたHTMLページを送信する必要なく、タグ付けされた HTMLコードを処理することが可能です。Webサーバーが開始されている必要もありません。

処理されるタグを格納しているデータを引数 inputData に渡します。この引数はBLOBまたはテキスト型の変数やフィールドです。テキスト型の使用が推奨されます(引数は2GB までのテキストを受け取ることができます)。

互換性に関する注意: 4D v12より、BLOB型の引数を使用すると、コマンドは自動でBlobに使用されている文字セットをMacRomanとして扱います。効率化のために、Unicodeモードで処理が実行されるテキスト型の引数を使用することを強く推奨します。

Webサーバーが起動しているかいないかに関わらず、すべての4Dタグがサポートされます (4DTEXT、4DHTML、4DSCRIPT、4DLOOP,など) 。

注: Webサーバー (Webプロセス) のフレームワーク以外で 4DINCLUDE タグを使用する場合:

- 4Dのローカルモードまたは4D Serverの場合、データベースストラクチャーファイルを格納しているフォルダーがデフォルトフォルダーです。
- 4Dのリモートモードの場合、4Dのアプリケーションを格納しているフォルダーがデフォルトフォルダーです。

コマンドの実行後引数 outputData には、引数 inputData に含まれる4Dタグが処理された結果が返されます。引数 inputData が4Dタグを含まない場合、引数 outputData の内容は引数 inputData の内容と一致します。
引数 outputData はフィールドまたは変数です。ただし引数 inputData と同じ型でなくてはなりません。

このコマンドは送信前に、4Dタグの処理結果をデータベースに格納することを可能にします。

またこのコマンドはWebサーバーの使用とは関係なく、4Dタグの解析を実行します。特に4D Internet Commandsを使用して、データベースにあるデータ処理の結果またはデータへの参照を格納しているHTMLフォーマットでe-mailのメッセージを 送る際に使用できます。

注: このコマンドはOn Web Authenticationデータベースメソッドを呼び出さなくなりました。

例題

この例題ではテンプレートドキュメントをロードし、そこに含まれるタグを処理し、ファイルとして書き出します:

```
C_BLOB($Blob_x)
C_BLOB($blob_out)
C_TEXT($inputText_t)
C_TEXT($outputText_t)

DOCUMENT TO BLOB("mytemplate.txt";$Blob_x)
$inputText_t:=BLOB to text($Blob_x;UTF8 Text without length)
PROCESS 4D TAGS($inputText_t;$outputText_t)
TEXT TO BLOB($outputText_t;$blob_out;UTF8 Text without length)
BLOB TO DOCUMENT($document;$blob_out)
```

SET ENVIRONMENT VARIABLE

SET ENVIRONMENT VARIABLE (varName ; varValue)

引数	型		説明
varName	文字	→	設定する変数の名前
varValue	文字	→	変数の値、またはデフォルト値をリセットする ""

説明

SET ENVIRONMENT VARIABLEコマンドを用いて、Mac OS XとWindowsで環境変数値を設定できます。このコマンドはLAUNCH EXTERNAL PROCESSコマンドと共に使用します。またPHP Executeコマンドとも動作します。

変数の名前をvarNameに、その値をvarValueに渡して定義します。

- 環境変数の概略リストと可能な値を取得するには、オペレーティングシステムのテクニカルドキュメントを参照してください。
- LAUNCH EXTERNAL PROCESSコマンドを用いて利用可能となる環境変数のリストをチェックするには、このコマンドのドキュメントを参照してください。このコンテキストでの使用に対して、3つの特定な環境変数が用意されています。
  _4D_OPTION_CURRENT_DIRECTORY: 開始する外部プロセスのカレントディレクトリを設定するために使用します。必ずディレクトリ(Mac OSではHFSタイプのシンタックス、WindowsではDOS) のパス名をvarValueに渡してください。
  _4D_OPTION_HIDE_CONSOLE (Windowsのみ): DOSコンソールのウィンドウを隠すために使用します。今コンソールを隠すには"True"をvarValueに渡します。コンソールを表示するには"False" を渡します。
  _4D_OPTION_BLOCKING_EXTERNAL_PROCESS: 非同期モードで外部プロセスを実行するために使用します。この場合他のアプリケーションをブロックしません。非同期実行を設定するには"False" を、同期実行を設定するには"True"をvarValue に渡してください (この変数にデフォルト値を設定することはできません)。
  これらの変数は、カレントプロセスでのLAUNCH EXTERNAL PROCESSの次の呼び出しから有効です。

例題

LAUNCH EXTERNAL PROCESSコマンドの例題を参照してください。

SET MACRO PARAMETER

SET MACRO PARAMETER (selector ; textParam)

引数	型	説明
selector	倍長整数	➡ 使用するセレクション
textParam	テキスト	➡ 送られたテキスト

説明

SET MACRO PARAMETERコマンドは、呼び出されたメソッドにテキストtextParamを挿入します。

テキストがメソッド内で選択された場合、引数selectorを使用して、テキストtextParamがすべてのメソッドテキストを置き換えるか、それとも選択されたテキストのみを置き換えるかを設定できます。セレクタには、テーマ"" に追加されている以下の定数を一つ渡します。

定数	型	値
Full method text	倍長整数	1
Highlighted method text	倍長整数	2

テキストが選択されていない場合、textParamがメソッドへ挿入されます。

注

GET MACRO PARAMETERとSET MACRO PARAMETERコマンドが正確に起動するには、新しい"バージョン" の属性が、以下のようにマクロ自体に記述されていなければなりません。

```
<macro name="MyMacro" version="2">
--- Text of macro ---
</macro>
```

例題

このマクロは新しいテキストを作成します。このテキストは呼び出しているメソッドへ返されます。

```
C_TEXT($input_text)
C_TEXT($output_text)
GET MACRO PARAMETER(Highlighted method text;$input_text)
 `選択されたテキストはテーブル、つまり "[Customers]" と仮定する
$output_text:=""
$output_text:=$output_text+Command name(47)+"("+$input_text+")" `すべて選択する ([Customers])
$output_text:=$output_text+"$i:="+Command name(76)+"("+$input_text+")" `セレクション ([Customers]) 内にある$i:=Records
SET MACRO PARAMETER(Highlighted method text;$output_text)
 `新しいコードで選択されたテキストを置き換える
```

データベースメソッド

- データベースメソッド
- On REST Authentication データベースメソッド New 14.0
- On Backup Shutdownデータベースメソッド
- On Backup Startupデータベースメソッド
- On Dropデータベースメソッド
- On Exitデータベースメソッド
- On Host Database Event データベースメソッド New 14.0
- On Server Close Connectionデータベースメソッド
- On Server Open Connectionデータベースメソッド
- On Server Shutdownデータベースメソッド
- On Server Startupデータベースメソッド
- On SQL Authenticationデータベースメソッド
- On Startupデータベースメソッド
- On System Event データベースメソッド
- On Web Authenticationデータベースメソッド
- On Web Connectionデータベースメソッド
- On Web Session Suspend データベースメソッド

データベースメソッド

データベースメソッドとは、セッションイベントが発生したときに4Dが自動で実行するメソッドです。

データベースメソッドを作成または開いて編集するには:

1. **エクスプローラ**ウィンドウを開く。
2. **メソッド**ページを選択する。
3. **データベースメソッド**テーマを拡げる。
4. メソッド名をダブルクリックする。

または:

1. メソッドを選択する。
2. enterキーまたはreturnキーを押す。

データベースメソッドは、他のメソッドと同じ方法で編集します。

データベースメソッドを他のメソッドから呼び出すことはできません。データベースメソッドは、作業セッション中のある時点で、4Dから自動で起動されます。以下の表は、データベースメソッドの実行の概要を示しています:

データベースメソッド	4D ローカル	4D Server	4D リモート
On Startup	○ (1回)	×	○ (1回)
On Exit	○ (1回)	×	○ (1回)
On Drop	○ (複数回)	×	○ (複数回)
On System Event	○ (複数回)	○ (複数回)	○ (複数回)
On Web Connection	○ (複数回)	○ (複数回)	○ (複数回)
On Web Authentication	○ (複数回)	○ (複数回)	○ (複数回)
On Web Session Suspend	○ (複数回)	○ (複数回)	○ (複数回)
On Backup Startup	○ (複数回)	○ (複数回)	○ (複数回)
On Backup Shutdown	○ (複数回)	○ (複数回)	○ (複数回)
On Server Startup	×	○ (1回)	×
On Server Shutdown	×	○ (1回)	×
On Server Open Connection	×	○ (複数回)	×
On Server Close Connection	×	○ (複数回)	×
On SQL Authentication	○ (複数回)	○ (複数回)	○ (複数回)
On REST Authentication	○ (複数回)	○ (複数回)	×
On Host Database Event	○ (複数回)	○ (複数回)	○ (複数回)

On REST Authentication データベースメソッド

$1, $2, $3 -> On REST Authentication データベースメソッド -> 戻り値

引数	型		説明
$1	テキスト	←	ユーザー名
$2	テキスト	←	パスワード
$3	ブール	←	True = ダイジェストモード False = ベーシックモード
戻り値	ブール	↪	True = リクエスト承認 False = リクエスト拒否

説明

On REST Authentication データベースメソッド はREST セッションを開くのを管理するための方法を提供します。このデータベースメソッドは主に Wakanda Server と4D v14との接続を設定するときにそれをフィルタリングするのが主な目的です。

Wakanda Server から ***mergeOutsideCatalog()*** メソッドを使用して REST セッションを開くリクエストが来ると(一般的なケース)、接続の識別子がリクエストのヘッダーに供給されます。続いて **On REST Authentication データベースメソッド** データベースメソッドが呼ばれこれらの識別子を評価します。4D データベースのユーザーのリストを使用することもできますし、独自の識別子のテーブルを使用することもできます。

重要: On REST Authentication データベースメソッド が定義される(つまり中にコードが記述される)と、4D はREST リクエストの管理をそちらに全て一任します。このとき、データベース設定のWeb/RESTページ内の「読み込み/書き出し」メニューで設定した内容は、無視されます(*Design Reference* マニュアルを参照して下さい)。

このデータベースメソッドは二つのテキスト型の引数($1 と $2)と一つのブール型の引数($3)を4Dから受け取り、ブール型の引数 $0 を返します。これらの引数は以下の様に宣言されている必要があります。

```
  //On REST Authentication データベースメソッド
 C_TEXT($1;$2)
 C_BOOLEAN($0;$3)
 ... // メソッドのコード
```

$1 には接続に使用したユーザー名が入り、$2 にはパスワードが入ります。

リクエストに使われるモードにより、パスワード ($2) は標準テキストまたはハッシュ値で受け取る事が可能です。このモードは $3 引数によって指定され、適切に処理することができます:

- パスワードが標準テキスト(ベーシックモード)である場合、$3 には**False** が渡されます。
- パスワードがハッシュ値(ダイジェストモード)である場合、$3 には **True** が渡されます。

REST接続リクエストがWakanda Serverから来るときは、パスワードは必ずハッシュ値で送られてきます。

リクエストがブラウザや Wakanda 以外の Web クライアントから送られてくる場合、デベロッパが責任を持って "username-4D" フィールドと "password-4D" フィールドを HTTP ヘッダーに含めることによってオリジナルのHTML/JavaScript ページからの認証を管理して下さい。この場合、パスワードは4D REST サーバーに標準テキストで送られてなければなりません(サードパーティからの干渉のリスクを避けるためにSSLを使用して下さい)。

REST 接続の識別子は、データベースメソッド内でチェックしなければなりません。通常、ユーザー独自のテーブルを使用して名前とパスワードをチェックします。もし識別子が有効であるなら、$0 に**True** を渡します。すると、リクエストが受理されます。4Dはこのリクエストを実行して結果をJSON形式で返します。
それ以外の場合は $0 に **False** を渡します。この場合、接続は拒否され、サーバーはリクエストの送信者へ認証をエラーを返します。

ユーザーがデータベースの4Dユーザーのリストの中に載っているとき、以下のコードによってパスワードを直接チェックすることができます:

```
 $0:=Validate password($1;$2;$3)
```

Validate password コマンドは拡張され、第一引数にユーザー名、第二引数にパスワードを渡し、任意の第三引数でパスワードがハッシュド形式で書かれているかどうかを指定できるようになりました。

4D データベースのものとは別の独自のユーザーリストを使用したい場合、そのユーザー達のパスワードを、Wakanda Server が **On REST Authentication データベースメソッド** データベースメソッドに接続リクエストを送る時のアルゴリズムと同じものを用いてハッシュ形式にて$2 引数に保存することができます。

この方法を使用してパスワードをハッシュする場合、以下の様に記述して下さい:

```
 $HashedPasswd :=Generate digest($ClearPasswd ;4D digest)
```

Generate digest コマンドにはハッシュアルゴリズムとして 4D digest を受け取れるようになりました。これは4Dのパスワードの内部管理で使用されているメソッドと対応しています。

例題 1

この例題ではパスワード "123"を使用する、4Dユーザーと合致しない "admin"というユーザーのみを受け入れる場合を考えます:

```
  //On REST Authentication database method
 C_TEXT($1;$2)
 C_BOOLEAN($0;$3)
  //$1: ユーザー
  //$2: パスワード
  //$3: ダイジェストモード
 If($1="admin")
    If($3)
       $0:=($2=Generate digest("123";4D digest))
    Else
       $0:=($2="123")
    End if
 Else
    $0:=False
 End if
```

例題 2

以下の **On REST Authentication データベースメソッド** の使用例は、接続リクエストが4D データベースのユーザーに保存されている二つの認証済みの Wakanda サーバーのどちらかから来ていることをチェックします:

```
 C_TEXT($1;$2)
 C_BOOLEAN($0)
 ON ERR CALL("REST_error")
 If($1="WAK1")|($1="WAK2")
    $0:=Validate password($1;$2;$3)
 Else
    $0:=False
End case
```

On Backup Shutdownデータベースメソッド

$1 -> On Backup Shutdownデータベースメソッド

引数	型		説明
$1	倍長整数	←	0 = バックアップは正しく実行された; 0以外の値 = エラー、ユーザーにより中断された、またはOn Backup Startupから返されたコード

On Backup Shutdownデータベースメソッドは、データベースのバックアップが終了するたびに呼び出されます。バックアップが終了する理由には、コピーの終了、 ユーザによる中断、そしてエラーがあります。
これはすべての4D環境: 4D (すべてのモード), 4D Server、4D Volume Desktopが統合されたコンパイル済みアプリケーションに当てはまります。

On Backup Shutdownデータベースメソッドを使用すると、バックアップが正常に実行されたかどうかを確認できます。バックアップが完了すると、このメソッド内の$1 引数にはバックアップのステータスを示す値が返されます:

- バックアップが正常に終了すると、$1には0が代入されます。
- バックアップがユーザにより中断されたり、エラーが発生した場合、$1には0以外が代入されます。
  - バックアップがOn Backup Startupデータベースメソッド ($0 # 0)により停止された場合、$1には$0 引数で返された値が代入されます。これにより、独自のエラー管理システムを実装できます。
  - エラーのためバックアップが停止した場合、エラーコードが$1に返されます。

  いずれの場合もGET BACKUP INFORMATIONコマンドを使用してエラーに関する情報を入手できます。

注: データベースメソッドで$1 引数 (倍長整数) を宣言しなければなりません:

```
C_LONGINT($1)
```

On Backup Startupデータベースメソッド

On Backup Startupデータベースメソッド -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	0 = バックアップの開始を許可する; 0以外の値 = バックアップの開始を許可しない

On Backup Startupデータベースメソッドは、データベースのバックアップを開始しようとするたびに呼び出されます（手動でのバックアップ、定期的自動バックアップ、または[BACKUP](#) コマンドによるバックアップ）。
これはすべての4D環境: 4D (すべてのモード), 4D Server、4D Volume Desktopが統合されたコンパイル済みアプリケーションに当てはまります。

On Backup Startupデータベースメソッドを使用すると、バックアップの開始を検証することができます。このメソッドからは、バックアップの許可または拒否を示す以下の値を引数$0にセットしてください:

- $0= 0：バックアップの開始を許可します。
- $0# 0：バックアップを拒否します。バックアップ処理はキャンセルされ、エラーが返されます。このエラーは、GET BACKUP INFORMATIONコマンドを使用して取得できます。

このデータベースメソッドを使用して、バックアップの実行条件を検証できます（ユーザ、前回のバックアップ日付など）。

Note: データベースメソッドで$0 引数 (倍長整数) を宣言しなければなりません:

```
C_LONGINT($0).
```

On Dropデータベースメソッド

On Dropデータベースメソッド
このコマンドは引数を必要としません

On Dropデータベースメソッドはローカルおよびリモートモードの4Dで使用できます。

このデータベースメソッドは、オブジェクトが4Dアプリケーションのフォームやウィンドウの外にドロップされると自動で実行されます。例えば:

- MDIウィンドウの空のエリア (Windows)
- Dockやシステムデスクトップの4Dアイコン (Mac OS)

Mac OSでは、データベースメソッドが呼び出されるためにはOption+Commandキーがドロップの際に押されていなければなりません。

デスクトップで4Dアプリケーションアイコンにドロップが行われると、アプリケーションが4D Volume Desktopにマージされている場合を除き、**On Dropデータベースメソッド**はアプリケーションが既に起動されている場合にのみ呼び出されます。マージされている場合、アプリケーションが起動されていなくてもデータベースメソッドは呼び出されます。これはカスタムのドキュメント署名を定義できることを意味します。

例題

この例題は、フォーム外側にドロップされた4D Sriteドキュメントを開きます:

```
  `On Drop database method
 droppedFile:=Get file from pasteboard(1)
 If(Position(".4W7";droppedFile)=Length(droppedFile)-3)
    externalArea:=Open external window(100;100;500;500;0;droppedFile;"_4D Write")
    WR OPEN DOCUMENT(externalArea;droppedFile)
 End if
```

On Exitデータベースメソッド

On Exitデータベースメソッド
このコマンドは引数を必要としません

On Exitデータベースメソッドは、データベースを終了すると一回呼び出されます。

このメソッドは、以下の4D環境で使用されます:

- ローカルモードの4D
- リモートモードの4D
- コンパイルし、4D VolumeDesktopを組み込んだ4Dアプリケーション

Note: **On Exitデータベースメソッド**は、4D Serverでは起動されません。

On Exitデータベースメソッドは4Dによって自動的に起動されます。プロジェクトメソッドとは異なり、プログラムからデータベースメソッドを呼び出すことはできません。しかしメソッドエディタから実行することはできます。またサブルーチンを使用できます。

データベースは、以下のうちいずれかが発生すると終了します:

- ユーザがデザインモードの**ファイル**メニュー**終了**メニューを、またはアプリケーションモードで終了標準アクション付きのメニューやボタンを選択した場合
- QUIT 4D コマンドへの呼び出しが実行された場合
- 4DプラグインからQUIT 4Dエントリポイントへの呼び出しが実行された場合

データベースの終了がどのような方法で行われたかに関わらず、4Dは以下のような処理を実行します:

- **On Exitデータベースメソッド**がない場合、4Dは実行プロセスそれぞれを区別せずに1つずつアボートします。ユーザがデータ入力を実行している場合には、レコードはキャンセルされ、保存されません。
- **On Exitデータベースメソッド**がある場合、4Dは新しく作成したローカルプロセスの中でこのメソッドの実行を開始します。したがって、このデータベースメソッドを使用し、インタープロセス間通信を通じて、(データ入力を)終了したり、処理の実行を中止したりしなければならないことを、他のプロセスに通知することができます。4Dは、いずれ終了するということに注意してください。**On Exitデータベースメソッド**は、必要なクリーンアップや終了の処理を実行することができますが、アプリケーションの終了を拒否できず、ある時点で終了することになります。

On Exitデータベースメソッドは、以下のような処理を実行するには最適です:

- データベースを開いた時に自動的に開始されたプロセスを停止する。
- 次回のセッションの始めにOn Startupデータベースメソッドで再利用するため、必要な環境設定や初期設定などの情報をローカルディスなどに保存する。
- データベースを終了する度に実行したい処理が他にあれば、それを実行する。

Note: **On Exitデータベースメソッド**がローカル/クライアントプロセスであり、データファイルにアクセスできないことを覚えておいてください。つまり、リモートモードで4Dを使用している時、**On Exitデータベースメソッド**で検索やソートを実行すると、4Dは "フリーズ" し、実際には終了しません。アプリケーションを終了するとき、リモートモードの4Dからデータにアクセスする必要があるならば、**On Exitデータベースメソッド**内から、新しいグローバルプロセスを作成してください。このグローバルプロセスはデータファイルにアクセスできます。この場合**On Exitデータベースメソッド**の終了前に、新しいプロセスが正しく終了されたことを (インタープロセス変数を使用するなどして) 確認してください。

例題

以下の例では、作業セッション中に発生する重要なイベントを追跡し、その説明を"Journal"という名前のテキストドキュメントに書き込むすべてのメソッドをカバーします。

- On Startupデータベースメソッドは、インタープロセス変数vbQuit4Dを初期化します。この変数は、すべての使用プロセスに、データベースが終了中であるか通知するために使用されます。またこのメソッドは、ジャーナルファイルが存在しない場合にはそれを作成します。

```
 ` On Startup データベースメソッド
C_TEXT(<>vtIPMessage)
C_BOOLEAN(<>vbQuit4D)
<>vbQuit4D:=False

If(Test path name("Journal")#Is a document)
   $vhDocRef:=Create document("Journal")
   If(OK=1)
      CLOSE DOCUMENT($vhDocRef)
   End if
End if
WRITE JOURNAL("セッションが開始されました")
```

- WRITE JOURNALプロジェクトメソッドは、他のメソッドからサブルーチンとして使用され、ジャーナルファイルに受け取った情報を書き込みます:

```
 ` WRITE JOURNAL プロジェクトメソッド
 ` WRITE JOURNAL ( Text )
 ` WRITE JOURNAL ( Event description )
C_TEXT($1)
C_TIME($vhDocRef)

While(Semaphore("$Journal"))
   DELAY PROCESS(Current process;1)
End while
$vhDocRef:=Append document("Journal")
If(OK=1)
   PROCESS PROPERTIES(Current process;$vsProcessName;$vlState;$vlElapsedTime;$vbVisible)
   SEND PACKET($vhDocRef;String(Current date)+Char(9)+String(Current time)+Char(9)
   +String(Current process)+Char(9)+$vsProcessName+Char(9)+$1+Char(13))
   CLOSE DOCUMENT($vhDocRef)
End if
CLEAR SEMAPHORE("$Journal")
```

ドキュメントが毎回開かれ閉じられることに注目してください。さらにドキュメントへのアクセス保護としてセマフォを利用していることにも注目してください。2つのプロセスがジャーナルファイルに同時にアクセスすることを防ぐためです。

- M_ADD_RECORDSプロジェクトメソッドは、アプリケーションモードで**レコード追加**メニュー項目が選択されると実行されます:

```
 ` M_ADD_RECORDS プロジェクトメソッド
MENU BAR(1)
REPEAT
   ADD RECORD([Table1];*)
   If(OK=1)
      WRITE JOURNAL("Table1へのレコード追加 #"+String(Record number([Table1])))
   End if
Until((OK=0)|<>vbQuit4D)
```

このメソッドは、ユーザが最後のデータ入力をキャンセルするか、またはデータベースを終了するまでループします。

- [Table 1]の入力フォームには、On Outside Callイベントの処理手順も含まれています。したがって、プロセスがデータ入力中であっても、ユーザは現在のデータ入力を保存するかまたは保存しないで、スムーズに終了できます:

```
 ` [Table1];"Input" フォームメソッド
Case of
   :(Form event=On Outside Call)
      If(<>vtIPMessage="QUIT")
         CONFIRM("このレコードに対する変更を保存しますか?")
         If(OK=1)
            ACCEPT
         Else
            CANCEL
         End if
      End if
End case
```

- M_QUITプロジェクトメソッドは、アプリケーションモードで**ファイル**メニューから**終了**が選択されると実行されます:

```
 ` M_QUIT プロジェクトメソッド
$vlProcessID:=New process("DO_QUIT";32*1024;"$DO_QUIT")
```

このメソッドには、ある仕掛けがあります。QUIT 4Dが呼び出されると、コマンドは即座に効果があります。したがって、呼び出しが発行されたプロセスは、データベースが実際に終了されるまでの間は"停止モード"になります。このプロセスは、データ入力が行われているプロセスの可能性があるため、QUIT 4Dの呼び出しは、この目的にだけ開始されるローカルプロセス内で実行されます。DO_QUITメソッドは、以下のようになります:

```
 ` DO_QUIT プロジェクトメソッド
CONFIRM("本当に終了しますか?")
If(OK=1)
   WRITE JOURNAL("データベース終了中")
   QUIT 4D
 ` QUIT 4Dは即座に効果があります。以下の行が実行されることはありません。
 ` ...
End if
```

- 最後に、すべての開いているユーザプロセスに対して"ただちに終了するよう"通知するOn Exitデータベースメソッドは、以下のようになります。このメソッドは、vbQuit4DをTrueに設定し、データ入力を実行しているユーザプロセスに、プロセス間メッセージを送信します:

```
 ` On Exit データベースメソッド
<>vbQuit4D:=True
REPEAT
   $vbDone:=True
   For($vlProcess;1;Count tasks)
      PROCESS PROPERTIES($vlProcess;$vsProcessName;$vlState;$vlElapsedTime;$vbVisible)
      If(((($vsProcessName="ML_@")|($vsProcessName="M_@"))) & ($vlState>=0))
         $vbDone:=False
         <>vtIPMessage:="QUIT"
         BRING TO FRONT($vlProcess)
         CALL PROCESS($vlProcess)
         $vhStart:=Current time
         REPEAT
            DELAY PROCESS(Current process;60)
         Until((Process state($vlProcess)<0)|((Current time-$vhStart)>=?00:01:00?))
      End if
   End for
Until($vbDone)
WRITE JOURNAL("セッション終了中")
```

Note: "ML_..."または"M_..."で始まる名前を持つプロセスは、**新規プロセス開始**プロパティを選択したメニューによって開始されます。この例では、そのようなプロセスは、**レコード追加**メニューが選択された時に開始されたプロセスです。

(Current time-$vhStart)>=?00:01:00?という判定式により、データベースメソッドは"他のプロセスを待っている"状態を終了します。他のプロセスがただちに反応しない場合には、ループを繰り返します。

・次に示しているのは、データベースによって作成されたジャーナルファイルの代表的な例です:

```
2/6/03 15:47:25 1 Main process セッションが開始されました
2/6/03 15:55:43 5 ML_1         Table1へのレコード追加 #23
2/6/03 15:55:46 5 ML_1         Table1へのレコード追加 #24
2/6/03 15:55:54 6 $DO_QUIT     データベース終了中
2/6/03 15:55:58 7 $xx          セッション終了中
```

Note: $xxという名前は、On Exitデータベースメソッドを実行するために4Dが開始したローカルプロセスの名前です。

On Host Database Event データベースメソッド

$1 -> On Host Database Event データベースメソッド

引数	型		説明
$1	倍長整数	←	イベントコード

Description

On Host Database Event データベースメソッド はデータベースが開いた時と閉じられた時に 4D コンポーネントにコードを実行させることが出来るようになりました。

注: セキュリティ上の理由から、このデータベースメソッドを使用可能にするためには、その実行をホストデータベースで明示的に許可する必要があります。詳細に関しては、*Design Reference* マニュアルを参照して下さい。

On Host Database Event データベースメソッド は、ホストデータベースのコンポーネントとして使用されているデータベースの中でのみ自動的に実行されます(ホストデータベースのプロパティリストで有効にされている必要があります)。このメソッドはホストデータベースの開閉に関するイベントが発生したときに呼び出されます。

イベントを扱うためには、$1 引数の値をメソッド内で調べて、"Database Events" テーマ内にある、以下の定数と比較する必要があります:

定数	型	値	コメント
On after host database exit	倍長整数	4	ホストデータベースの On Exitデータベースメソッド データベースメソッドが実行を終了したところです。
On after host database startup	倍長整数	2	ホストデーターベースの On Startupデータベースメソッド データベースメソッドが実行を終了したところです。
On before host database exit	倍長整数	3	ホストデータベースは閉じられようとしているところです。ホストデータベースの On Exitデータベースメソッド データベースメソッドはまだ呼び出されていない状態です。 ホストデータベースの On Exitデータベースメソッド データベースは、コンポーネントの On Host Database Event データベースメソッド データベースメソッドが実行されている間は呼び出されません。
On before host database startup	倍長整数	1	ホストデータベースはちょうど開かれたところです。ホストデータベースの On Startupデータベースメソッド データベースメソッドはまだ呼び出されていません。 On Startupデータベースメソッド データベースメソッドは、On Host Database Event データベースメソッド データベースメソッドがコンポーネント内で実行されている間は呼び出されません。

このコマンドにより、4D コンポーネントはホストデータベースのオペレーションに関連したプリファレンスやユーザー情報を読み込んだり保存したりすることが出来ます。

例題

On Host Database Event の典型例を紹介します:

```
  // On Host Database Event データベースメソッド
 C_LONGINT($1)
 Case of
    :($1=On before host database startup)
  // この部分にホストデータベースの "On Startup" データベースメソッドの前に実行したいコードを記述します
    :($1=On after host database startup)
  // この部分にホストデータベースの "On Startup" データベースメソッドの後に実行したいコードを記述します
    :($1=On before host database exit)
  // この部分にホストデータベースの "On Exit" データベースメソッドの前に実行したいコードを記述します
    :($1=On after host database exit)
  // この部分にホストデータベースの "On Exit" データベースメソッドの後に実行したいコードを記述します
 End case
```

On Server Close Connectionデータベースメソッド

$1, $2, $3 -> On Server Close Connectionデータベースメソッド

引数	型		説明
$1	倍長整数	⇐	ユーザーを識別するために4D Serverが内部的に使用するユーザーID
$2	倍長整数	⇐	接続を識別するために4D Serverが内部的に使用する接続ID
$3	倍長整数	⇐	廃止: 常に0が返されますが、宣言はしなくてはなりません。

説明

On Server Close Connectionデータベースメソッドは、4Dクライアントプロセスが終了するたびに、サーバマシン上で一度呼び出されます。

On Server Open Connectionデータベースメソッドの場合と同様に、4D Server は**On Server Close Connectionデータベースメソッド**に3つの倍長整数タイプの引数を渡しますが、結果は求めません。

したがって、このメソッドでは3つの引数を倍長整数として明示的に宣言しなくてはなりません:

```
C_LONGINT($1;$2;$3)
```

次の表は、このデータベースメソッドに渡される3つの引数が示す情報を表わしています:

引数	説明
$1	4D Serverがユーザを識別するために内部的に使用するユーザID番号
$2	4D Serverが接続を識別するために内部的に使用する接続ID番号
$3	廃止: 常に0が渡されますが、宣言は必要

On Server Close Connectionデータベースメソッドは、On Server Open Connectionデータベースメソッドと対をなすメソッドです。4Dクライアントプロセスについての詳細は、このデータベースメソッドの説明を参照してください。

例題

On Server Open Connectionデータベースメソッドの例題参照

On Server Open Connectionデータベースメソッド

$1, $2, $3 -> On Server Open Connectionデータベースメソッド -> $0
引数型　　説明
$1　倍長整数 ← ユーザーを識別するために4D Serverが内部的に使用するユーザーID
$2　倍長整数 ← 接続を識別するために4D Serverが内部的に使用する接続ID
$3　倍長整数 ← 廃止: 常に0が返されますが、宣言はしなくてはなりません。
$0　倍長整数 → 0または省略時 = 接続を受け入れる、0以外 = 接続を拒否する

On Server Open Connection データベースメソッドはいつ呼び出されるか

On Server Open Connectionデータベースメソッドは、4Dリモートワークステーションが接続プロセスを開始するたびに、サーバーマシン上で一度、呼び出されます。4D Server 以外の4D 環境では**On Server Open Connectionデータベースメソッド**が起動されることはありません。

On Server Open Connectionデータベースメソッドは以下のときに呼び出されます:

- リモート4Dが接続した (アプリケーションプロセスが開始するため)
- リモート4Dがデザインモードを開く (デザインプロセスが開始するため)
- リモート4Dが、サーバー上でコオペラティブプロセスの作成を必要とする* (プロセス名が"$"で始まらない) グローバルプロセスを開始した。このプロセスはNew processコマンド、メニューコマンド、またはメソッド実行ダイアログボックスを使用して作成されます

リモート4Dでは、いずれの場合にも3つのプロセスが開始されます (クライアントマシン上に1つ、また必要に応じてサーバマシン上に2つ) 。クライアントマシンでは、プロセスでコードが実行され、4D Serverに要求が送られます。サーバマシンでは、**4Dクライアントプロセス**はクライアントプロセスのためのデータベース環境 (ユーザプロセスのためのカレントセレクションやレコードのロック等) を管理し、クライアントマシン上で実行中のプロセスから送られた要求に対して応答を返します。**4Dクライアントデータベースプロセス**は対応する4Dクライアントプロセスのモニタを担当します。

(*) 4D v13より最適化のため、サーバープロセス (データベースエンジンにアクセスするためのプリエンプティブプロセスとランゲージアクセスのためのコオペラティブプロセス) はクライアント側のコードを実行する際必要な時にだけ作成されるようになりました。以下は新規クライアントプロセスを実行する4Dコードの詳細です:

```
  // グローバルプロセスが開始されるがこの時点では
  //サーバー上にはプロセスは作成されない
 CREATE RECORD([Table_1])
 [Table_1])field1_1:="Hello world"
 SAVE RECORD([Table_1]) // この時点でサーバーにプリエンプティブプロセスが作成される
 $serverTime:=Current time(*) // ここでサーバー上にコオペラティブプロセスが作成され
  // On Server Open Connectionが呼び出される
```

重要: Web接続およびSQL接続は**On Server Open Connectionデータベースメソッド**を起動しません。Webブラウザーが4D Server に接続する場合は On Web Authenticationデータベースメソッド (あれば) とOn Web Connectionデータベースメソッドが起動されます。4D ServerがSQLクエリを受信すると、On SQL Authenticationデータベースメソッドが (あれば) 呼び出されます。詳細については、4D Language Referenceマニュアルのデータベースメソッドに関する説明を参照してください。

重要: ストアドプロシージャの開始時には、**On Server Open Connectionデータベースメソッド**は起動されません。ストアドプロシージャ はサーバプロセスであり、4Dクライアントプロセスではありません。ストアドプロシージャはサーバマシン上でコードを実行しますが、4Dクライアント (または他のクライアント) と4D Server によってやり取りされる要求に対して応答を返すことはありません。

On Server Open Connection データベースメソッドはどのように呼び出されるか

On Server Open Connectionデータベースメソッド は4D Serverマシン上で、このメソッドを呼び出しを引き起こした4Dクライアントプロセス内で実行されます。

例えばリモート4Dが4D Server インタープリタデータベースに接続すると、そのクライアント用のユーザプロセスとデザインプロセス、クライアント登録プロセスが (デフォルトで) 開始されます。したがって On Server Open Connectionデータベースメソッド は3回実行されます。つまり1回目はアプリケーションプロセス内で、2回目はクライアント登録プロセス内で、3回目はデザインプロセス内で実行されます。3つのプロセスがそれぞれサーバマシン上で開始される6番目と7番目と8番目のプロセスである場合に、**On Server Open Connectionデータベースメソッド**内から Current process を呼び出すと、Current process は1回目には6を、2回目には7を、3回目に8を返します。

On Server Open Connectionデータベースメソッド はサーバマシン上で実行されることに注意してください。このデータベースメソッドは、クライアント 側で実行中のプロセスとは無関係に、サーバマシン上で実行中の4Dクライアントプロセス内で実行されます。また、このメソッドが起動された時点では、4D クライアントプロセスにはまだ名前が付いていません (この時点では、PROCESS PROPERTIESは4Dクライアントプロセスの名前を返しません)。

On Server Open Connectionデータベースメソッド は、クライアント側で実行中のプロセスのプロセス変数テーブルにアクセスしません。このテーブルはサーバマシンではなく、クライアントマシンに存在します。

On Server Open Connectionデータベースメソッド がプロセス変数にアクセスすると、4Dクライアントプロセス用に動的にプロセス変数テーブルが作成され、プライベートに使用されます。

4D Serverは **On Server Open Connectionデータベースメソッド** に3つの倍長整数タイプの引数を渡し、倍長整数タイプの結果を求めます。したがってこのメソッドでは3つの引数と戻り値を倍長整数として明示的に宣言しなくてはなりません:

```
 C_LONGINT($0;$1;$2;$3)
```

$0に値を返さず、その結果変数を未定義のままにするかまたはゼロに初期化した場合、4D Server はデータベースメソッドが接続を受け付けたものとみなします。接続を受け付けない場合、$0にヌルではない値を返します。

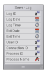
次の表はこのデータベースメソッドに渡される3つの引数が示す情報を表わしています:

引数	説明
$1	4D Serverがユーザを識別するために内部的に使用するユーザID番号
$2	4D Serverが接続を識別するために内部的に使用する接続ID番号
$3	廃止: 常に0が渡されますが、宣言は必要

これらのID番号は、例えば4Dコマンドに渡す引数のように、情報ソースとして直接使用することはできません。しかしこれらのID番号は**On Server Open Connectionデータベースメソッド**とOn Server Close Connectionデータベースメソッドとの間で、4Dクライアントプロセスを一意に識別するために利用できます。4D Server セッションのどの時点でも、これらの値の組み合わせはユニークです。インタープロセス配列やテーブルにこの情報を格納することによって、2つのデータベー スメソッド間で情報をやり取りできます。この節の最後に示された例では、2つのデータベースメソッドがこの情報を使用して、テーブルの同一レコードに接続 の開始と終了の日付と時間を格納しています。

例題 1

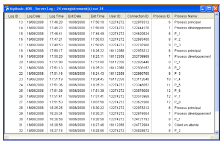
次の例は**On Server Open Connectionデータベースメソッド**と**On Server Close Connection データベースメソッド**を使用して、データベースへの接続ログを管理する方法を示しています。**[Server Log]**テーブル (下図) は接続処理の記録を取るために使用されています:



このテーブルに格納される情報は、次の**On Server Open Connectionデータベースメソッド**と**On Server Close Connection データベースメソッド**によって管理されます:

```
  ` On Server Open Connection データベースメソッド
 C_LONGINT($0;$1;$2;$3)
  ` [Server Log] レコード作成
 CREATE RECORD([Server Log])
 [Server Log]Log ID:=Sequence number([Server Log])
  ` 接続日付と時間を保存
 [Server Log]Log Date:=Current date
 [Server Log]Log Time:=Current time
  ` 接続情報を保存
 [Server Log]User ID:=$1
 [Server Log]Connection ID:=$2
 SAVE RECORD([Server Log])
  ` エラーなしを返すと接続が続行される


 $0:=0 ` On Server Close Connection データベースメソッド
 C_LONGINT($1;$2;$3)
  ` [Server Log] レコードを取得
 QUERY([Server Log];[Server Log]User ID=$1;*)
 QUERY([Server Log];&;[Server Log]Connection ID=$2)
  ` 終了日付と時間を保存
 [Server Log]Exit Date:=Current date
 [Server Log]Exit Time:=Current time
  ` プロセス情報を保存
 [Server Log]Process ID:=Current process
 PROCESS PROPERTIES([Server Log]Process ID;$vsProcName;$vlProcState;$vlProcTime)
 [Server Log]Process Name:=$vsProcName
 SAVE RECORD([Server Log])
```

下図は[Server Log]に登録されたレコードで、いくつかのリモート接続を示しています:



Log ID :	Log Date :	Log Time	Exit Date :	Exit Time	User ID :	Connection ID	Process ID	Process Name :
13	16/06/2008	17:46:20	16/06/2008	17:50:10	12274272	122978312	6	Process principal
14	16/06/2008	17:46:23	16/06/2008	17:50:09	12274272	122444176	7	Process développement
15	16/06/2008	17:46:41	16/06/2008	17:46:49	12274272	124620824	8	P_1
16	16/06/2008	17:47:21	16/06/2008	17:50:03	12274272	122683400	8	P_2
17	16/06/2008	17:49:53	16/06/2008	17:50:05	12274272	122797960	9	P_3
18	16/06/2008	17:50:17	16/06/2008	18:25:22	16112358	122978312	6	Process principal
19	16/06/2008	17:50:20	16/06/2008	18:25:11	16112358	252709968	7	Process développement
20	16/06/2008	17:51:08	16/06/2008	17:51:08	16112358	122826440	8	P_1
21	16/06/2008	17:51:13	16/06/2008	18:25:21	16112358	122939152	8	P_2
22	16/06/2008	17:51:16	16/06/2008	18:24:43	16112358	122960760	9	P_3
23	16/06/2008	17:51:19	16/06/2008	18:24:45	16112358	123112040	10	P_4
24	16/06/2008	17:51:36	16/06/2008	18:25:21	12274272	123346952	11	P_5
25	16/06/2008	17:51:39	16/06/2008	17:51:39	12274272	123575008	12	P_6
26	16/06/2008	17:51:41	16/06/2008	17:51:41	12274272	123575968	12	P_7
27	16/06/2008	17:51:53	16/06/2008	18:07:56	12274272	123621968	12	P_8
28	16/06/2008	18:25:25	16/06/2008	18:30:22	12274272	122978312	6	Process principal
29	16/06/2008	18:25:34	16/06/2008	18:30:21	12274272	122879504	7	Process développement
30	16/06/2008	18:26:58	16/06/2008	18:26:58	12274272	124727792	8	P_1
31	16/06/2008	18:26:58	16/06/2008	18:27:46	16112358	124772984	9	Client en attente
32	16/06/2008	18:27:16	16/06/2008	18:28:06	12274272	124828872	8	P_2

例題 2

以下の例題は午前2時から4時の間の接続を拒否します。

```
  ` On Server Open Connection データベースメソッド
 C_LONGINT($0;$1;$2;$3)

 If((?02:00:00?<=Current time)&(Current time<?04:00:00?))
    $0:=22000
 Else
    $0:=0
 End if
```

On Server Shutdownデータベースメソッド

On Server Shutdownデータベースメソッド
このコマンドは引数を必要としません

On Server Shutdownデータベースメソッドは、カレントのデータベースが4D Server上で閉じられるときに、サーバマシン上で一度呼び出されます。4D Server以外の4D 環境では**On Server Shutdownデータベースメソッド**が起動されることはありません。

サーバ上のカレントデータベースを閉じるには、サーバ上で**データベースを閉じる...** メニューコマンドを使用します。また**4D Serverを終了** メニューコマンドを選択したり、サーバ上で実行されるストアドプロシージャ内でQUIT 4D コマンドを呼び出すこともできます。

データベースの終了が開始されると、4D は次の動作を実行します:

- **On Server Shutdownデータベースメソッド**がない場合、4D Server は実行中の各プロセスを区別なく1 つずつアボートします。
- **On Server Shutdownデータベースメソッド**がある場合、4D Server は新しく作成されたローカルプロセス内でこのメソッドの実行を開始します。したがって、このデータベースメソッドを使用し、プロセス間通信を介して、他のプロセスに対し、実行を停止するよう通知することができます。結局は、4D Server が終了するという点に注意してください**On Server Shutdownデータベースメソッド**では、片付けたり、クローズする操作をすべて実行することができますが、終了を拒否することはできないため、いずれかの時点で終了することになります。

On Server Shutdownデータベースメソッドは次の事柄を行うのに最適です:

- データベースが開かれた時に自動的に起動されたストアードプロシージャを停止する
- 次のセッションの始めに**On Server Shutdownデータベースメソッド**で再使用するために、初期設定や各種設定を（ディスク上にローカルに）保存する
- データベースが終了するたびに自動的に実行させたいその他の動作を実行する

警告: On Server Shutdownデータベースメソッドを使用してストアドプロシージャをクローズする場合、サーバは (ストアドプロシージャではなく)**On Server Shutdownデータベースメソッド**が実行されると終了することに留意してください。この時点でストアドプロシージャが起動されていると、それらはキルされます。
このため、サーバによりキルされる前に、ストアドプロシージャが完全に実行されたことを確認したい場合、**On Server Shutdownデータベースメソッド**はストアドプロシージャに対し実行を終了しなければならないことを通知して (例えばインタープロセス変数を使用)、そして終了を待つようにするべきです (x秒のループや他のインタープロセス変数を使用)。

リモートの4Dがサーバへの接続を停止する時に、クライアントマシン上で自動的にコードを実行させたい場合には、On Exitデータベースメソッドを使用してください。

On Server Startupデータベースメソッド

On Server Startupデータベースメソッド
このコマンドは引数を必要としません

On Server Startupデータベースメソッドは、4D Server でデータベースを開くと、サーバマシン上で一度呼び出されます。4D Server 以外の4D環境で**On Server Startupデータベースメソッド**が起動されることはありません。

On Server Startupデータベースメソッドは次の事柄を行うのに最適です:

- 4D Server セッション全体を通して使用するインタープロセス変数を初期化する
- データベースが開かれる時に自動でストアドプロシージャを開始する
- 前の4D Serverセッション中に保存された初期設定や各種設定をロードする
- 明示的にQUIT 4Dを呼び出すことによって、(システムリソースが見つからない等) 条件が満たされていない場合にデータベースを開けないようにする
- データベースが開かれるたびに自動的に実行させたいその他の動作を実行する

リモート4Dがサーバに接続する時に、クライアントマシン上で自動的にコードを実行するにはOn Startupデータベースメソッドを使用します。

Note: On Server Startup データベースメソッドはアトミックに実行されます。つまりこのメソッドの実行が終了するまで、リモート4Dは接続を行うことができません。

On SQL Authenticationデータベースメソッド

$1, $2, $3 -> On SQL Authenticationデータベースメソッド -> 戻り値

引数	型	説明
$1	テキスト	← ユーザー名
$2	テキスト	← パスワード
$3	テキスト	← (オプション) リクエスト送信元クライアントのIPアドレス
戻り値	ブール	↩ True = リクエストを受け入れる、False = リクエストを拒否する

On SQL Authenticationデータベースメソッドは4Dに統合されたSQLサーバへ送られたリクエストを選別します。この選別は、名前とパスワード、そしてユーザのIPアドレス (オプション) に基づいて実行されます。開発者は独自のユーザーテーブルや、4Dのユーザーテーブルを使用して、接続を識別できます。接続を認証したら、CHANGE CURRENT USER コマンドを呼び出して、4Dのデータベース内のリクエストへのアクセスをコントロールしなければなりません。

On SQL Authenticationデータベースメソッドが存在する場合、4Dまたは4D ServerのSQLサーバに外部からSQL接続が行われると、自動的にこのメソッドが呼び出されます。4Dユーザを管理する内部システムは起動しません。データベースメソッドが$0にTrueを返しかつ、CHANGE CURRENT USERコマンドの実行が成功した場合のみ、接続が受け入れられます。これらの条件を満たさない場合リクエストは拒否されます。

Note: SQL LOGIN(SQL_INTERNAL;$user;$password)ステートメントは内部接続となるため、On SQL Authenticationデータベースメソッドを呼び出しません。

データベースメソッドは最大3つのテキストタイプの引数を4Dより受け取り、$0にブール値を返します。以下はこれらの引数の説明です。

引数	型	説明
$1	テキスト	ユーザ名
$2	テキスト	パスワード
$3	テキスト	(オプション) リクエスト元のクライアントのIPアドレス
$0	ブール	True = リクエストを許可, False = リクエストを拒否

以下のようにこれらの引数を宣言しなければなりません:

```
  ` On Web Authentication データベースメソッド

 C_TEXT($1;$2;$3)
 C_BOOLEAN($0)
  ` メソッドコード
```

標準テキストとして、パスワード($2)を受け取ります。

On SQL AuthenticationデータベースメソッドでSQL接続の識別子を確認します。例えば、ユーザのカスタムテーブルを使用して名前とパスワードをチェックします。識別子が有効な場合は、$0にTrueを返して接続を受け入れます。その他の場合は$0に、Falseを返して接続が拒否されます。

Note: On SQL Authenticationデータベースメソッドが存在しない場合、4Dの統合されたユーザ管理システムを使用して、接続を決定します (有効になっている場合、つまりDesignerにパスワードが割り当てられている場合)。このシステムが起動していない場合、ユーザはDesignerアクセス権 (フリーアクセス) で接続されます。

$0にTrueを渡す場合、リクエストを受け入れ、ユーザのためにSQLのセッションを開くためには、**On SQL Authenticationデータベースメソッド**でCHANGE CURRENT USER コマンドを呼び出し、その実行が成功しなければなりません。

CHANGE CURRENT USER コマンドは、仮想の認証システムを実行するために使用されます。この認証システムには、2つの利点があります。1つは接続動作をコントロールできること。もう1つは4DのSQLセッションで接続の識別子を外部から見えないようにします。

以下の例で**On SQL Authenticationデータベースメソッド**は、接続リクエストが内部ネットワークからのものであることを確認し、識別子を検証、SQLセッションへのアクセス権に"sql_user" ユーザを割り当てます。

```
 C_TEXT($1;$2;$3)
 C_BOOLEAN($0)
  `$1: user
  `$2: password
  `($3: IP address of client)
 ON ERR CALL("SQL_error")
 If(checkInternalIP($3))
  `checkInternalIPメソッドはIPアドレスが内部のものか確認します。
    If($1="victor") & ($2="hugo")
       CHANGE CURRENT USER("sql_user";"")
       If(OK=1)
          $0:=True
       Else
          $0:=False
       End if
    Else
       $0:=False
    End if
 Else
    $0:=False
 End if
```

On Startupデータベースメソッド

On Startupデータベースメソッド
このコマンドは引数を必要としません

On Startupデータベースメソッドは、データベースを開くと1度呼び出されます。

このメソッドは、以下の4D環境で使用されます:

- ローカルモードの4D
- リモートモードの4D (4D Serverから接続を許可された後、クライアント側で)
- コンパイルし、4D VolumeDesktopを組み込んだ4Dアプリケーション

Note: **On Startupデータベースメソッド**は、4D Serverでは起動されません。

On Startupデータベースメソッドは、4Dによって自動的に起動されます。プロジェクトメソッドとは異なり、プログラムからデータベースメソッドを呼び出すことはできません。しかしメソッドエディタから実行することはできます。またサブルーチンを使用できます。

On Startupデータベースメソッドは、以下のような処理を実行するのに最適です:

- 作業セッション全体で使用するインタープロセス変数を初期化する。
- データベースを開いた時にプロセスを自動的に開始する。
- 以前の作業セッション中に保存された環境設定や初期設定をロードする。
- (システムリソースがない等) 条件が一致しない場合、QUIT 4Dを明示的に呼び出してデータベースの開始を拒否する。
- データベースを開く度に自動的に実行したい他の動作を実行する。

しかしながら**On Startupデータベースメソッド**から印刷ジョブを起動することは推奨しません。

例題

On Exitデータベースメソッドの例題を参照

On System Event データベースメソッド

$1 -> On System Event データベースメソッド

引数	型		説明
$1	倍長整数	⇐	イベントコード

説明

On System Event データベースメソッド はシステムイベントが発生するたびに呼び出されます。すべてのモードの4D、4D Server、4D Volume Desktopが統合されたコンパイル済みアプリケーションなど、すべての4D環境で有効です。

イベントを処理するために、メソッドの中で$1引数をテストし、Database Eventsテーマの以下の定数と比較しなければなりません:

定数	型	値	コメント
On after host database exit	倍長整数	4	ホストデータベースの On Exitデータベースメソッド データベースメソッドが実行を終了したところです。
On after host database startup	倍長整数	2	ホストデーターベースの On Startupデータベースメソッド データベースメソッドが実行を終了したところです。
On application background move	倍長整数	1	4Dアプリケーションがバックグランドに移動した
On application foreground move	倍長整数	2	4Dアプリケーションが最前面に移動した
On before host database exit	倍長整数	3	ホストデータベースは閉じられようとしているところです。ホストデータベースの On Exitデータベースメソッド データベースメソッドはまだ呼び出されていない状態です。 ホストデータベースの On Exitデータベースメソッド データベースは、コンポーネントの On Host Database Event データベースメソッド データベースメソッドが実行されている間は呼び出されません。
On before host database startup	倍長整数	1	ホストデータベースはちょうど開かれたところです。ホストデータベースの On Startupデータベースメソッド データベースメソッドはまだ呼び出されていません。 On Startupデータベースメソッド データベースメソッドは、On Host Database Event データベースメソッド データベースメソッドがコンポーネント内で実行されている間は呼び出されません。

これらのイベントは4Dアプリケーションのレベルが変わったときに生成されます。例えば:

- 4Dあるいは他のアプリケーションのウィンドウがクリックされた。
- **Alt+Tab** (Windows) や **Command+Tab** (Mac OS) キーボードショートカットでアプリケーションが選択された。
- Dockの**隠す**コマンドが選択された (Mac OS)。
- Dockやタスクバーでアプリケーションアイコンがクリックされた。
- メインウィンドウで最小化ボタンがクリックされた (Windows)。

データベースメソッド内では$1を倍長引数型で宣言しなければなりません。データベースメソッドの構造は以下のようになります:

```
 // On System Event データベースメソッド

C_LONGINT($1)
Case of
   :($1=On application background move)
 // 処理を行う
   :($1=On application foreground move)
 // 処理を行う
End case
```

On Web Authenticationデータベースメソッド

$1, $2, $3, $4, $5, $6 -> On Web Authenticationデータベースメソッド -> 戻り値

引数	型		説明
$1	テキスト	←	URL
$2	テキスト	←	HTTPヘッダー + HTTPボディ(32 KBまで)
$3	テキスト	←	Webクライアント (ブラウザー) のIPアドレス
$4	テキスト	←	サーバーのIPアドレス
$5	テキスト	←	ユーザー名
$6	テキスト	←	パスワード
戻り値	ブール	↩	True = リクエストを受け入れる, False = リクエストを拒否する

説明

On Web AuthenticationデータベースメソッドはWebサーバーエンジンへのアクセス管理を担当します。このデータベースメソッドは、Webブラウザーからのリクエストがサーバー上の4Dメソッド (**4DACTION** URLや**4DSCRIPT** などを使用して呼び出されるメソッド) の実行を必要とするとき、4Dから呼ばれます。

このメソッドは6つのテキスト引数$1, $2, $3, $4, $5, $6を受け取り、ブール値を$0に返します。これらの引数の意味は以下のとおりです:

引数	型	説明
$1	テキスト	URL
$2	テキスト	HTTPヘッダー + HTTPボディ (32 KBまで)
$3	テキスト	Webクライアント (ブラウザー) のIPアドレス
$4	テキスト	サーバーのIPアドレス
$5	テキスト	ユーザー名
$6	テキスト	パスワード
$0	ブール	True = リクエストを受け入れる, False = リクエストを拒否する

これらの引数を以下のように宣言しなければなりません:

```
  ` On Web Authentication データベースメソッド
C_TEXT($1;$2;$3;$4;$5;$6)
C_BOOLEAN($0)
  ` メソッドコード
```

注: On Web Authenticationデータベースメソッドのすべての引数が必ず値を受け取るわけではありません。データベースメソッドが受け取る情報はデータベース設定ダイアログボックスでの設定により異なります (接続セキュリティ参照)。

- **URL**

  最初の引数 ($1) はユーザーがWebブラウザーのアドレスエリアに入力した**URL** (からホストのアドレスを取り除いたもの) です。

  イントラネット接続の場合をみてみましょう。4D WebサーバーのIPアドレスを123.4.567.89とします。以下の表はWebブラウザーに入力された**URL**により、$1が受け取る値を示しています:

Webブラウザーのアドレスに入力された値	$1の値
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers/Add	/Customers/Add
123.4.567.89/Do_This/If_OK/Do_That	/Do_This/If_OK/Do_That

- **HTTPリクエストのヘッダーとボディ**

  2番目の引数 ($2) はWebブラウザーから送信されたHTTPリクエストのヘッダーとボディです。この情報は**On Web Authenticationデータベースメソッド**にそのまま渡されることに留意してください。その内容は接続を試みたWebブラウザーの仕様により異なります。

  アプリケーションでこの情報を使用するには、開発者がヘッダーとボディを解析しなければなりません。

  注:
  - パフォーマンスの理由から、このデータのサイズは32KBで切り取られます。
  - この引数に関する詳細はOn Web Connectionデータベースメソッドの説明を参照してください。

- **WebクライアントのIPアドレス**

  $3 引数はブラウザマシンのIPアドレスを受け取ります。この情報を使用して、イントラネットアクセスとインターネットアクセスを区別できます。

- **サーバーIPアドレス**

  $4 引数はWebサーバを呼び出すために使用されたIPアドレスを受け取ります。4Dバージョン6.5以降マルチホーミングがサポートされ、複数のIPアドレスを持つマシンで使用できるようになりました。詳細はQR DELETE COLUMNを参照してください。

- **ユーザー名とパスワード**

  $5 と $6 引数は、ブラウザーが表示する標準の認証ダイアログにユーザーが入力したユーザー名とパスワードを受け取ります。このダイアログはデータベース設定でパスワード管理オプションが指定されていると、接続のたびに表示されます (接続セキュリティ参照)。

注: ブラウザーから送信されたユーザー名が4Dに存在する場合、$6 引数 (ユーザーパスワード) はセキュリティのため渡されません。

- **$0 引数On Web Authenticationデータベースメソッド**はブール値を$0に返します:
  - $0=**True**: 接続を受け入れる
  - $0=**False**: 接続を受け入れない

On Web Connectionデータベースメソッドは、接続が**On Web Authenticationデータベースメソッド**により受け入れられた時にのみ実行されます。

警告: $0に値が設定されないか、$0が**On Web Authenticationデータベースメソッド**内で定義されていない場合、接続は受け入れるものとされ、On Web Connectionデータベースメソッドが実行されます。

注:

- **On Web Authenticationデータベースメソッド**内でインターフェース要素を呼び出さないでください (ALERT, DIALOG等)。そうでなければメソッドの実行が中断され、接続は拒否されます。処理中にエラーが発生した場合も同じことが言えます。
- メソッドプロパティダイアログオックスのオプション"4DタグとURL (4DACTION...) で利用可"を使用して、プロジェクトメソッドごとに、**4DACTION** や **4DSCRIPT**にからメソッドを実行させないようにできます。この点に関する詳細は接続セキュリティを参照してください。

On Web Authentication データベースメソッドの呼び出し

On Web Authenticationデータベースメソッドは、リクエストや処理が4Dメソッドの実行を必要とするとき自動で呼び出されます。またWebサーバーが無効なスタティックURLを受け取ったときにも呼び出されます (例えばリクエストされたスタティックページが存在しない場合)。

まとめると**On Web Authenticationデータベースメソッド**は以下のケースで呼び出されます:

- 4Dが /4DACTION/ で始まるURLを受信したとき。
- 4Dが /4DCGI/ で始まるURLを受信したとき。
- 4Dが /4DSYNC/ で始まるURLを受信したとき。
- 4Dが、存在しないスタティックページをリクエストするURLを受信したとき。
- 4Dがセミダイナミックページで 4DSCRIPT タグを処理するとき。
- 4D がセミダイナミックページでメソッドに基づく 4DLOOP タグを処理するとき。

互換性に関する注意: このデータベースメソッドは4Dが/4DMETHOD/ で始まるURLを受信したときにも実行されます。このURLは廃止予定であり、互換性の目的で保持されています。

有効なスタティックページをリクエストするURLを受信したとき、**On Web Authenticationデータベースメソッド**は呼び出されないことに注意してください。

例題 1

BASIC認証モードの**On Web Authenticationデータベースメソッド**の例題:

```
  `On Web Authentication データベースメソッド
C_TEXT($1;$2;$3;$4;$5;$6)
C_BOOLEAN($0)

C_TEXT($user;$password;$BrowserIP;$ServerIP)
C_BOOLEAN($4Duser)
ARRAY TEXT($users;0)
ARRAY LONGINT($nums;0)
C_LONGINT($upos)

$0:=False

$user:=$5
$password:=$6
$BrowserIP:=$3
$ServerIP:=$4

  `セキュリティのため@を含むユーザー名とパスワードは拒否
If(WithWildcard($user)|WithWildcard($password))
   $0:=False
  `WithWildcard メソッドは後述
Else
  `4Dユーザーかチェック
   GET USER LIST($users;$nums)
   $upos:=Find in array($users;$user)
   If($upos >0)
      $4Duser:=Not(Is user deleted($nums{$upos}))
   Else
      $4Duser:=False
   End if

   If(Not($4Duser))
  `4Dに定義されたユーザーでない場合、Webusersテーブルを検索
      QUERY([WebUsers];[WebUsers]User=$user;*)
      QUERY([WebUsers]; & [WebUsers]Password=$password)
      $0:=(Records in selection([WebUsers])=1)
   Else
      $0:=True
   End if
End if
  `イントラネット接続か?
If(Substring($BrowserIP;1;7)#"192.100.")
   $0:=False
End if
```

例題 2

DIGESTモードのの例題:

```
  `On Web Authentication データベースメソッド
C_TEXT($1;$2;$5;$6;$3;$4)
C_TEXT($user)
C_BOOLEAN($0)
$0:=False
$user:=$5
  `セキュリティのため@を含むユーザ名とパスワードは拒否
If(WithWildcard($user))
   $0:=False
  `WithWildcard メソッドは後述
Else
   QUERY([WebUsers];[WebUsers]User=$user)
   If(OK=1)
      $0:=Validate Digest Web Password($user;[WebUsers]password)
   Else
      $0:=False `User does not exist
   End if
End if
```

WithWildcard メソッド:

```
  `WithWildcard Method
  `WithWildcard ( 文字列 ) -> ブール
  `WithWildcard ( Name ) -> @を含む

C_INTEGER($i)
C_BOOLEAN($0)
C_TEXT($1)

$0:=False
For($i;1;Length($1))
   If(Character code(Substring($1;$i;1))=Character code("@"))
      $0:=True
   End if
End for
```

On Web Connectionデータベースメソッド

$1, $2, $3, $4, $5, $6 -> On Web Connectionデータベースメソッド

引数	型		説明
$1	テキスト	⇐	URL
$2	テキスト	⇐	HTTPヘッダー + HTTPボディ (32 KBまで)
$3	テキスト	⇐	Webクライアント (ブラウザー) のIPアドレス
$4	テキスト	⇐	サーバーのIPアドレス
$5	テキスト	⇐	ユーザー名
$6	テキスト	⇐	パスワード

On Web Connectionデータベースメソッドは以下のケースで呼び出されます:

- Webサーバが /4DCGI/ から始まるURLを受信した。
- Webサーバが無効なリクエストを受信した。

詳細な情報は、後述の"On Web Connection データベースメソッド呼び出し" の段落を参照してください。

互換性に関する注意: コンテキストモードでコンテキストが作成されたときもデータベースメソッドは呼び出されます。コンテキストモードは廃止予定のモードで、変換されたデータベースで利用できます。

データベースがWebサーバとして公開され、リクエストは事前にOn Web Authenticationデータベースメソッドで受け入れられていなければなりません(存在する場合)。

On Web Connectionデータベースメソッドは6つのテキスト引数を受け取ります。これらの引数の内容は以下のとおりです:

引数	型	説明
$1	テキスト	URL
$2	テキスト	HTTPヘッダ + HTTPボディ (32 KBまで)
$3	テキスト	Webクライアント (ブラウザ) のIPアドレス
$4	テキスト	サーバのIPアドレス
$5	テキスト	ユーザ名
$6	テキスト	パスワード

これらの引数を以下のように宣言しなければなりません:

```
  // On Web Connection データベースメソッド
 C_TEXT($1;$2;$3;$4;$5;$6)
  // メソッドコード
```

- **URL**

  最初の引数 ($1) は、ユーザがWebブラウザのアドレスエリアに入力した**URL**からホストのアドレスを取り除いたものです。
  イントラネット接続の場合をみてみましょう。4D WebサーバのIPアドレスを123.4.567.89とします。以下の表はWebブラウザに入力された**URL**に対して、$1が受け取る値を示しています:

Webブラウザのアドレスに入力された値	$1の値
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers/Add	/Customers/Add
123.4.567.89/Do_This/If_OK/Do_That	/Do_This/If_OK/Do_That

  この引数は必要に応じて自由に利用できます。4Dは単にURLのホスト部より後の部分を$1に渡します。
  例えば値 "/Customers/Add" が "直接新規レコードを[Customers] テーブルに追加する"ということを意味するというような、オリジナルのルールを作成できます。Webユーザにデータベースを公開し、利用可能な値やブックマークを提供できます。アプリケーションの異なる部分へのショートカットを提供できます。このようにして、Webユーザはデータベースに接続するたびにナビゲーションを通過することなく、素早くWebサイトのリソースにアクセスできます。

警告: 以前のセッションで作成されたブックマークでデータベースに再入力されることを防ぐため、4Dは標準の4D URLに対応するURLをすべてキャッチします。

- **HTTPリクエストのヘッダとボディ**

  ２番目の引数 ($2) はWebブラウザから送信されたHTTPリクエストのヘッダとボディです。この情報は**On Web Connectionデータベースメソッド**にそのまま渡されることに留意してください。その内容は接続を試みたWebブラウザの仕様により異なります。
  Mac OS上のSafariでは、以下のようなヘッダを受け取るでしょう (一部省略):

  ```
  GET /favicon.ico HTTP/1.1
  Referer: http://123.45.67.89/4dcgi/test
  User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X; fr-fr) AppleWebKit/523.10.3 (KHTML, like Gecko)
  Version/3.0.4 Safari/523.10
  Cache-Control: max-age=0
  Accept: */*
  Accept-Language: ja-jp
  Accept-Encoding: gzip, deflate
  Connection: keep-alive
  Host: 123.45.67.89
  ```

  Windows上のMicrosoft Internet Explorer 8では、以下のようなヘッダを受け取るでしょう:

  ```
  GET / HTTP/1.1
  Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml, image/pjpeg,
  application/x-ms-xbap, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
  Accept-Language: ja-JP
  User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET
  CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C)
  Accept-Encoding: gzip, deflate
  Host: 123.45.67.89
  Connection: Keep-Alive
  ```

  アプリケーションでこの情報を使用するには、開発者がヘッダとボディを解析しなければなりません。
  注: パフォーマンスの理由から、このデータのサイズは32KBで切り取られます。

- **WebクライアントのIPアドレス**

  $3 引数はブラウザマシンのIPアドレスを受け取ります。この情報を使用して、イントラネットアクセスとインターネットアクセスを区別できます。

- **サーバIPアドレス**

  $4 引数はWebサーバを呼び出すために使用されたIPアドレスを受け取ります。4Dバージョン6.5以降マルチホーミングがサポートされ、複数のIPアドレスを持つマシンで使用できるようになりました。詳細はWebサーバー設定を参照してください。

- **ユーザ名とパスワード**

  $5 と $6 引数は、ブラウザが表示する標準の認証ダイアログにユーザが入力したユーザ名とパスワードを受け取ります。このダイアログは環境設定でパスワード管理オプションが指定されていると、接続のたびに表示されます (接続セキュリティ参照)。

注: ブラウザから送信されたユーザ名が4Dに存在する場合、$6 引数 (ユーザパスワード) はセキュリティのため渡されません。

On Web Connection データベースメソッドの呼び出し

は4DCGI URLまたはカスタマイズされたコマンドURLを使用したWebサーバーへのアクセスのエントリポイントとして使用できます。

警告: インタフェース要素を表示する4D コマンド (ALERT, DIALOG...) を呼び出すと、メソッド処理が終了します。

は以下のケースで呼び出されます:

- 4Dが */4DCGI/* URLを受け取ったとき。$1に */4DCGI/<action>* が渡されて、データベースメソッドが呼び出されます。
- *<path>/<file>*タイプのURLで存在しないWebページが呼び出されたとき。データベースメソッドにそのURLが渡されて呼び出されます (*)。
- *<file>/* タイプのURLでWebページが呼び出され、デフォルトのホームページが設定されていないとき。データベースメソッドにそのURLが渡されて呼び出されます (*)。

(*) 特定のケースでは、$1に渡されるURLはスラッシュ"/"で始まりません。

On Web Session Suspend データベースメソッド

On Web Session Suspend データベースメソッド
このコマンドは引数を必要としません

On Web Session Suspend データベースメソッド はWebセッションが閉じられる直前に、4D Webサーバーから呼び出されます。4Dは以下のような場合にWebセッション (セッションを管理するWebプロセス) を閉じます:

- セッションを管理するWebプロセス数の最大値 (デフォルトで100、WEB SET OPTIONコマンドで変更可能) に達している状態で、さらに新しいWebセッションを作成する必要があるとき (4Dは一番古いWebセッションプロセスを自動で破棄します)
- セッションプロセスのタイムアウトに達したとき (デフォルトで480分 = 8時間、WEB SET OPTIONコマンドで変更可能)
- WEB CLOSE SESSIONコマンドが呼び出された場合

このデータベースメソッドが呼び出された時点で、セッションのコンテキスト (プロセス変数の値やカレントセレクション) は有効です。そのセッションに関連するデータ (変数の値やセレクション) を退避し、後で同じcookie値でリクエストを受信したときにそれらを再利用することができます。

On Web Session Suspend データベースメソッドの例題はWebセッション管理を参照してください。

データ入力

- ACCEPT
- ADD RECORD
- CANCEL
- DIALOG
- MODIFY RECORD
- Old
- ADD SUBRECORD
- Modified
- MODIFY SUBRECORD

ACCEPT

ACCEPT
このコマンドは引数を必要としません

説明

ACCEPT コマンドは以下の目的で、フォームメソッドまたはオブジェクトメソッド（またはサブルーチン）で使用されます：

- ADD RECORD, MODIFY RECORD, ADD SUBRECORD, MODIFY SUBRECORDを使用して開始されたレコードやサブレコードのデータ新規あるいは更新入力を受け入れる。
- DIALOG コマンドで表示されたフォームを受け入れる。
- DISPLAY SELECTION や MODIFY SELECTIONでレコードセレクションを表示したフォームを閉じる。

ACCEPT はユーザがEnterキーを押したのと同じ動作をします。フォームが受け入れられると、OKシステム変数に1が設定されます。

ACCEPT は一般的にメニューコマンド選択結果として実行されます。またACCEPT は"アクションなし"ボタンのオブジェクトメソッドで使用されます。

またOpen window コマンドのオプションのクローズボックスメソッドでもしばしば使用されます。ウィンドウ上にコントロールメニューボックスがあれば、コントロールメニューボックスをクリックまたは閉じるメニューコマンドが選択されたときに実行されるメソッド中でACCEPT または CANCEL を呼び出すことができます。

ACCEPT を実行待ちのキューに追加することはできません。イベントへのレスポンスとしてメソッド中で2つのACCEPT コマンドを実行しても、1回実行したのと同じ効果しかありません。

ADD RECORD

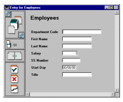
ADD RECORD ({aTable}{;}{*})
aTable テーブル→データ入力に使用するテーブル, または 省略した場合デフォルトテーブル
* →スクロールバーを隠す

説明

ADD RECORDコマンドは、データベースのテーブルaTableまたはaTableが省略された場合デフォルトテーブルに、ユーザが新規レコードを追加できるようにします。

ADD RECORDは新しいレコードを作成、それをカレントプロセスのカレントレコードとし、カレントの入力フォームを表示します。アプリケーションモードにおいて、ユーザが新しいレコードを受け入れると、新しいレコードがカレントセレクションにある唯一のレコードになります。

次の図は、典型的なデータ入力フォームです。



このフォームはこのプロセスの最前面のウインドウに表示されます。ウインドウには、スクロールバーとサイズボックスがあります。オプションの * 引数を指定すると、スクロールバーやサイズボックスのないウインドウを描画します。

ADD RECORDは、ユーザがレコードを受け入れるか取り消すまでフォームを表示します。ユーザが複数のレコードを追加する場合は、新しいレコードごとに1回ずつコマンドを実行しなければなりません。

ユーザが保存ボタンをクリック、またはenterキーを押す、またはACCEPTコマンドが実行されると、レコードが保存されます。

ユーザがキャンセルボタンをクリック、またはキャンセルキーコンビネーション (WindowsではCtrl-ピリオド、Mac OSではCommand-ピリオド) を押す、またはCANCELコマンドが実行されると、レコードは保存されません。

ADD RECORDの呼び出し後、システム変数OKにはレコードが受け入れられると1が、キャンセルされると0が設定されます。

Note: キャンセルされた場合でも、レコードはメモリ上に残されたままとなります。カレントレコードポインタが変更される前にSAVE RECORDコマンドを実行すれば、レコードは保存されます。

例題 1

次の例は、データベースに新しいレコードを追加する際によく使われるループです:

```
FORM SET INPUT([Customers];"Std Input") ` [Customers] テーブルの入力フォームを設定
REPEAT ` ユーザがキャンセルするまでループ
   ADD RECORD([Customers];*) ` [Customers] テーブルにレコードを追加
Until(OK=0) ` ユーザがキャンセルするまで
```

例題 2

次の例は、顧客データを検索し、その検索結果により、2つのステートメントうちの1つを実行します。顧客が全く見つからない場合、ユーザはADD RECORDコマンドで新しい顧客を追加できます。少なくとも1つの顧客レコードが見つかった場合は、MODIFY RECORDにより最初のレコードが表示され、このレコードを修正できます:

```
READ WRITE([Customers])
FORM SET INPUT([Customers];"Input") ` 入力フォームを設定
vlCustNum:=Num(Request("顧客番号を入力:")) ` 顧客番号を取得
If(OK=1)
   QUERY([Customers];[Customers]CustNo=vlCustNum) ` 顧客を検索
   If(Records in selection([Customers])=0) ` 顧客が見つからなければ...
      ADD RECORD([Customers]) ` 新規に顧客を追加
   Else
      If(Not(Locked([Customers])))
         MODIFY RECORD([Customers]) ` レコードを更新
         UNLOAD RECORD([Customers])
      Else
         ALERT("レコードは現在使用中です。")
      End if
   End if
End if
```

システム変数およびセット

レコードを受け入れるとOKシステム変数が1に、キャンセルすると0に設定されます。OKシステム変数はレコードが受け入れられたかキャンセルされた後に設定されます。

CANCEL

CANCEL
このコマンドは引数を必要としません

説明

CANCEL コマンドは以下の目的で、フォームあるいはオブジェクトメソッド (またはそこから呼ばれるサブルーチンで) 使用されます:

- ADD RECORD, MODIFY RECORD, ADD SUBRECORD, MODIFY SUBRECORDを使用して開始されたレコードやサブレコードのデータ新規あるいは更新入力をキャンセルする。
- DIALOG コマンドで表示されたフォームをキャンセルする。
- DISPLAY SELECTION や MODIFY SELECTIONでレコードセレクションを表示したフォームを閉じる。
- Print form コマンドで印刷されようとしているフォームの印刷をキャンセルする (後述)。

データ入力のコンテキストで、CANCEL はキャンセルキー (**Esc**) を押したのと同じ動作をします。

CANCEL は一般的にメニューコマンド選択結果として実行されます。またCANCEL は"アクションなし"ボタンのオブジェクトメソッドで使用されます。

またOpen window コマンドのオプションのクローズボックスメソッドでもしばしば使用されます。ウィンドウ上にコントロールメニューボックスがあれば、コントロールメニューボックスをクリックまたは閉じるメニューコマンドが選択されたときに実行されるメソッド中でCANCEL または CANCEL を呼び出すことができます。

CANCEL を実行待ちのキューに追加することはできません。イベントへのレスポンスとしてメソッド中で2つのCANCEL コマンドを実行しても、1回実行したのと同じ効果しかありません。

最後に、このコマンドはPrint form コマンド使用時にOn Printing Detail フォームイベントで使用できます。このコンテキストでは、CANCEL コマンドは印刷しようとしていたフォームの印刷を一時的に停止し、次のページから再開します。このメカニズムは印刷スペースがなくなったときや、ページブレークが必要な時に使用できます。

Note: この処理はすべての印刷待ちフォームをキャンセルするPAGE BREAK(*) コマンドとは動作が異なります。

例題

SET PRINT MARKERの例題を参照

システム変数およびセット

CANCELコマンドが実行される (フォームや印刷がキャンセルされた) とシステム変数OKは0に設定されます。

DIALOG

DIALOG ({aTable ;} form {; *})

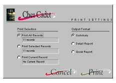
引数	型	説明
aTable	テーブル	⇒フォームの属するテーブルまたは 省略した場合デフォルトテーブルまたは プロジェクトフォームを使用
form	文字	⇒ダイアログとして表示するテーブルフォームまたは プロジェクトフォーム
*	演算子	⇒同じプロセスを使用

説明

DIALOGコマンドはユーザに対してフォームformを表示します。このコマンドは、変数を用いてユーザから情報を取得したり、処理を実行する際のオプションなど情報をユーザに表示するために、使用されます。

Open windowコマンドで作成したモーダルウィンドウにフォームを表示するのは一般的な使用方法です。

典型的なダイアログの例を次に示します:



表示したり取得しなければならない情報がより複雑で、ALERT, CONFIRM, または Requestで処理できない場合、これらのコマンドの代わりに**DIALOG**を使用してください。

注: 4D環境設定の互換性ページにあるオプションを使用して、ダイアログボックスのフィールドへのデータ入力を禁止し、変数にのみ入力可とすることができます。この制約は、以前のバージョンの4Dでの動作に相当します。

ADD RECORDやMODIFY RECORDと異なり、**DIALOG**はカレント入力フォームを使用しません。form引数で使用するフォーム (プロジェクトフォームまたはテーブルフォーム) を指定しなければなりません。また、ボタンを省略した場合でもデフォルトボタンパネルは表示されません。この場合**Esc**キーを使用してのみフォームを終了できます。

ユーザが保存ボタンをクリック、またはテンキー上の"enter"キーを押す、またはACCEPTコマンドが実行された場合、ダイアログが受け入れられます。ダイアログの受け入れが保存を行わないことを覚えておいてください。ダイアログにフィールドが含まれる場合、更新されたデータを保存するには明示的にSAVE RECORDコマンドを呼ばなければなりません。

ユーザがキャンセルボタンをクリック、または**Esc**キーを押す、またはCANCELコマンドが実行された場合には、ダイアログはキャンセルされます。

オプションの * 引数を渡すと、フォームはカレントプロセスで最後に開かれたウィンドウにロードされ、コマンドはフォームをスクリーン上でアクティブにしたまま実行を終了します。
このフォームはユーザアクションに対し通常通り反応し、フォームに関連する4Dコード (オブジェクトメソッドやフォームメソッド) がCANCELやACCEPTコマンドを呼び出すと閉じられます。カレントプロセスが終了すると、この方法で作成されたフォームは、CANCELコマンドが呼ばれたのと同じ方法で自動で閉じられます。この方法は特に、別のプロセスを起動する必要なく、ドキュメントと一緒にフローティングパレットを表示できる点で有用です。

DIALOGの呼び出し後、システム変数OKにはダイアログが受け入れられると1が、キャンセルされると0が設定されます。

例題 1

以下の例題は、検索条件を指定するためにDIALOGを使用します。変数vName と vState が置かれたフォームが表示され、ユーザは検索条件を入力できます:

```
Open window(10;40;370;220) ` モーダルウィンドウを表示
DIALOG("Search Dialog") ` カスタム検索ダイアログを表示
CLOSE WINDOW ` モーダルウィンドウは必要ない
If(OK=1) ` ダイアログが受け入れられれば
   QUERY([Company];[Company]Name=vName;*)
   QUERY([Company];&;[Company]State=vState)
End if
```

例題 2

以下の例題はツールパレットを作成するために使用できます:

```
  `ツールパレットの表示
$palette_window:=Open form window("tools";Palette form window)
DIALOG("tools";*) `即座にコントロールを返す
  `メインのドキュメントウィンドウを表示
$document_window:=Open form window("doc";Plain form window)
DIALOG("doc")
```

システム変数およびセット

DIALOGの呼び出し後ダイアログが受け入れられればOKに1が、キャンセルされれば0が設定されます。

MODIFY RECORD

MODIFY RECORD ({aTable}{;}{*})

aTable	テーブル	→ データ入力に使用するテーブル, または 省略した場合デフォルトテーブル
*		→ スクロールバーを隠す

説明

MODIFY RECORDコマンドは、aTableテーブルまたはaTable引数を省略した場合デフォルトテーブルのカレントレコードを修正するために使用します。MODIFY RECORDは、カレントプロセスにレコードがまだロードされていない場合にレコードをロードし、カレント入力フォームにレコードを表示します。カレントレコードがなければ、MODIFY RECORDは何も行いません。またMODIFY RECORDはカレントセレクションに影響を与えません。

フォームは現在のプロセスの最前面ウインドウに表示されます。ウインドウにはスクロールバーとサイズボックスがあります。オプションの * 引数を指定すると、スクロールバーやサイズボックスのないウインドウを表示します。

MODIFY RECORDを使用するには、必ずカレントレコードは読み込み/書き込み可能であり、ロックされていてはいけません。フォームにカレントセレクションのレコード内を移動するためのボタンがある場合、ユーザはこれらのボタンをクリックして、レコードを修正した後、他のレコードへ移動することができます。

ユーザが保存ボタンをクリック、またはenterキーを押す、またはACCEPTコマンドが実行されると、レコードが保存されます。

ユーザがキャンセルボタンをクリック、またはキャンセルキーコンビネーション (WindowsではCtrl-ピリオド、Mac OSではCommand-ピリオド) を押す、またはCANCELコマンドが実行されると、レコードは保存されません。キャンセルされた場合でもレコードはメモリ上に残されていて、カレントレコードポインタが変更される前にSAVE RECORDを実行すれば、レコードを保存できます。

MODIFY RECORDの呼び出し後、システム変数OKにはレコードが受け入れられると1が、キャンセルされると0が設定されます。

MODIFY RECORDを使用したが、ユーザがレコードのデータを変更しなかった場合、レコードは更新されたとは扱われず、レコードを受け入れても保存処理は行われません。変数の変更、チェックボックスのチェック、ラジオボタンの選択はレコードの更新とはみなされません。データ入力またはメソッドでフィールドデータの更新が行われた場合のみ、レコードの保存が実行されます。

例題

ADD RECORDコマンドの例題参照。

システム変数およびセット

レコードが受け入れらると、システム変数OKに1を代入します。キャンセルされた場合は、システム変数OKに0を代入します。OKシステム変数はレコードが受け入れられたかキャンセルされた後に設定されます。

Old

Old (aField) -> 戻り値

引数	型		説明
aField	フィールド	→	元の値を取得するフィールド
戻り値	式	↩	元のフィールド値

説明

Old コマンドは、プログラムにより値が代入されたり、データ登録で修正される前にaFieldに納められていた値を返します。

テーブルのカレントレコードを移動するたびに、4Dは新しいカレントレコードがメモリーにロードされた時点での複製された"イメージ"をメモリ上に作成し、管理します。レコードを修正する際には、レコードの実際のイメージを使います。複製イメージではありません。カレントレコードを移動すると、このイメージは破棄されます。

Oldはこの複製イメージの値を返します。すなわち、既存のレコードに対しては、ディスク上に保存されているフィールドの値を返すということです。新しく作成されたレコードの場合、Oldはそのフィールドタイプに応じた空の値を返します。例えば、aFieldが文字フィールドの場合、Oldは空の文字列を、数値フィールドならOldは0を返します。

Oldは、aFieldがメソッドまたはデータ入力時にユーザによって修正された場合にも機能します。

Oldは、すべてのフィールドタイプに適用できます。

フィールドの元の値を復元するには、Oldから返された値を割り当てます。

Note: 技術的な理由により、ピクチャやBLOBタイプのフィールドの場合、Oldから返される式を直接他のコマンドの引数としては利用できません。他の引数を経由する必要があります。例えば:

```
  `Do NOT write (causes a syntax error):
 $size :=BLOB size(Old([theTable]theBlob)) `INCORRECT
  `Write:
 $oldBLOB:=Old([theTable]theBlob)
 $size :=BLOB size($oldBLOB) `CORRECT
```

ADD SUBRECORD

ADD SUBRECORD (subtable ; form {; *})

subtable	サブテーブル	→	データ入力に使用するサブテーブル
form	文字	→	データ入力に使用するフォーム
*		→	スクロールバーを隠す

互換性に関する注意

バージョン11の4Dよりサブテーブルはサポートされていません。変換されたデータベースでは、互換メカニズムによりこのコマンドの機能が確保されています。しかし、サブテーブルを標準のリレートテーブルに置き換えることを強くお勧めします。

説明

ADD SUBRECORDコマンドは、formを使用して、subtableに新しいサブレコードをユーザが追加することを可能にします。ADD SUBRECORDはメモリ上に新しいサブレコードを作成し、それをカレントサブレコードとし、そしてformを表示します。親テーブルのカレントレコードは必ず存在しなければなりません。プロセスにカレント親レコードが存在しない場合、ADD SUBRECORDは何も行いません。フォームは、必ずsubtableに属さなければなりません。

ユーザが保存ボタンをクリック、またはテンキー上の"enter"キーを押す、またはACCEPTコマンドが実行されると、サブレコードはメモリに保持されます。サブレコードの追加後、このサブレコードを保存するためには、親レコードを明示的に保存しなければなりません。

サブレコードは、ユーザがキャンセルボタンがクリック、またはキャンセルキーコンビネーション (WindowsではCtrl-ピリオド、Mac OSではCommand-ピリオド) を押す、またはCANCELコマンドが実行された場合、サブレコードは保存されません。

ADD SUBRECORDの呼び出し後、システム変数OKにはサブレコードが受け入れられると1が、キャンセルされると0が設定されます。

フォームはプロセスの最前面のウインドウに表示されます。ウインドウには、スクロールバーとサイズボックスがあります。オプションの * 引数を指定すると、スクロールバーやサイズボックスのないウインドウを描画します。

例題

以下の例はメソッドの一部です。これは従業員レコードに新しい子どもサブレコードを追加します。子どものデータは、[Employees]Childrenサブテーブルに格納されます。サブレコードを保存するには、[Employees]レコードを保存する必要があるという点に注意してください:

```
ADD SUBRECORD([Employees]Children;"Add Child")
If(OK=1) ` ユーザがサブレコードを受け入れたら
   SAVE RECORD([Employees]) ` 従業員レコードを保存する
End if
```

システム変数およびセット

サブレコードを受け入れるとOKシステム変数は1に、キャンセルすると0に設定されます。

Modified

Modified (aField) -> 戻り値

引数	型		説明
aField	フィールド	→	テストするフィールド
戻り値	ブール	↩	フィールドに新しい値が代入されていればTrue, そうでなければFalse

互換性に関する注意

この関数は互換性のために保持されています。この関数は、バージョン6の4Dで廃止されたイベントの管理方法 (Before, Duringその他のコマンド参照) に基づいています。現在は、Form eventコマンドを使用し、`On Data Change`イベントが返されるかどうかをチェックする方法の使用を強く推奨します。

説明

Modifiedはデータ入力中、プログラムを使用してfieldに値が代入されていたり、値が編集された場合に、Trueを返します。Modifiedコマンドはフォームメソッド(またはフォームメソッドから呼ばれたサブルーチン)で使用されなければなりません。

このコマンドは同じ実行サイクル内でのみ意味のある値を返します。特に以前のDuring実行サイクルに対応するフォームイベントでは、Falseに設定されます。

データ入力時には、(元の値が変更されたかどうかに関わらず)ユーザがフィールドを編集した後別のフィールドへ移動するか、コントロールをクリックすると、フィールドが更新されたとみなされます。tabキーでフィールドを移動しただけでは、ModifiedはTrueにならない点に注意してください。ModifiedがTrueになるためには、フィールドが編集されなければなりません。

メソッドの実行時には、フィールドに値が割り当てられると(異なる値かどうかに関係なく)、フィールドが編集されたものと解釈されます。

Note: Modifiedは、PUSH RECORDとPOP RECORDコマンド実行後は、常にTrueを返します。

いずれの場合でも、フィールドの値が実際に変更されたかどうかを調べるには、Old コマンドを使用します。

Note: Modifiedはあらゆるタイプのフィールドに対して適用できますが、このコマンドをOld コマンドと組み合わせて使用する場合には、Old コマンドの制約に注意してください。詳細についてはOld コマンドの説明を参照してください。

データ入力時には、フォームメソッドでModifiedを使用するよりも、オブジェクトメソッドで処理を実行する方が簡単です。フィールドが修正される度にOn Data Changeイベントがオブジェクトメソッドに送信されるので、オブジェクトメソッドの利用はフォームメソッドでModifiedを使用したのと同じ意味を持ちます。

Note: 処理を正しく実行するため、Modified コマンドはフォームメソッドまたは、フォームメソッドから呼び出されるメソッド内でのみ使用します。

例題 1

次の例は、[Orders]Quantity フィールドや[Orders]Price フィールドが変更されたかどうかを判定します。どちらかが変更されると、[Orders]Total フィールドを再計算します。

```
If((Modified([Orders]Quantity)|(Modified([Orders]Price))
   [Orders]Total :=[Orders]Quantity*[Orders]Price
End if
```

2番目の行をサブルーチンにして、[Orders]Quantity フィールドと[Orders]Price フィールドのオブジェクトメソッドのOn Data Changeフォームイベントでそのサブルーチンを呼び出しても、同じ結果となります。

例題 2

[anyTable]テーブルのレコードを選択し、次に[anyTable]Important fieldフィールドが修正される可能性がある複数のサブルーチンを呼び出しますが、これらのメソッドはレコードの保存を行いません。メインのメソッドの終わりで、Modified コマンドを使用してレコードを保存する必要があるかどうかを調べています:

```
 ` レコードがカレントレコードとして選択済みです
 ` サブルーチンを使用して処理を行います
DO SOMETHING
DO SOMETHING ELSE
DO NOT FORGET TO DO THAT
 ` ...
 ` レコードを保存する必要があるか検証します
If(Modified([anyTable]Important field))
   SAVE RECORD([anyTable])
End if
```

⚙ MODIFY SUBRECORD

MODIFY SUBRECORD (subtable ; form {; *})

subtable	サブテーブル	⇒データを登録するサブテーブル
form	文字	⇒データ入力に使用するフォーム
*		⇒スクロールバーを隠す

互換性に関する注意

バージョン11の4Dより、サブテーブルはサポートされていません。変換されたデータベースでは、互換メカニズムによりこのコマンドの機能が確保されています。しかし、サブテーブルを標準のリレートテーブルに置き換えることを強くお勧めします。

説明

MODIFY SUBRECORD コマンドは、formを使用して、修正するsubtableのカレントサブレコードを表示します。フォームは必ずsubtableに属していなければなりません。

親テーブルのカレントレコードは必ず存在していなけれればなりません。プロセスにカレント親レコードが存在しない場合、MODIFY SUBRECORDコマンドは何も行いません。さらに、カレントサブレコードが存在しない場合にも、MODIFY SUBRECORDコマンドは何も行いません。

サブレコードの修正後、このサブレコードを保存するためには、親レコードを保存しなければなりません。

MODIFY SUBRECORDコマンドの実行後、システム変数OKにはサブレコードの変更が受け入れられると1が、キャンセルされると0が代入されます。

フォームはこのプロセスの最前面のウインドウに表示されます。ウインドウには、スクロールバーとサイズボックスがあります。オプションの * 引数を指定すると、スクロールバーやサイズボックスのないウインドウを表示します。

システム変数およびセット

レコードが受け入れられた場合、システム変数OKに1を代入します。キャンセルされた場合は、システム変数OKに0を代入します。

テーブル

- Current default table
- Current form table
- DEFAULT TABLE
- NO DEFAULT TABLE

Current default table

Current default table -> 戻り値

引数	型		説明
戻り値	ポインター	⮐	デフォルトテーブルへのポインタ

説明

Current default table は、カレントプロセスに対してDEFAULT TABLEコマンドで最後に指定されたテーブルのポインタを返します。

例題

デフォルトテーブルが設定されているものとして、以下のコードはカレントデフォルトテーブルの名前をウィンドウタイトルにセットします。

```
SET WINDOW TITLE(Table name(Current default table))
```

Current form table

Current form table -> 戻り値

引数	型	説明
戻り値	ポインター ➡	現在表示されているフォームが属すテーブルへのポインタ

説明

Current form tableコマンドは、カレントプロセスで表示または印刷されているフォームが属するテーブルのポインタを返します。

以下の場合では、関数はNilを返します。

- カレントプロセスに表示または印刷されているフォームがない。
- カレントフォームがプロジェクトフォームである。

カレントプロセスで複数のウィンドウが開いている(最後に開かれたウィンドウがカレントアクティブウィンドウになる) 場合、このコマンドはアクティブウィンドウにフォームが表示されているテーブルへのポインタを返します。

現在表示されているフォームがサブフォームエリア用の詳細フォームである場合、ユーザがデータ入力中であり、ダブルクリック可能なサブフォームエリアのレコードまたはサブレコードをダブルクリックしたことを意味します。この場合には、コマンドは以下を返します:

- サブフォームがテーブルを表示している場合、サブフォームエリアに表示されたそのテーブルへのポインタ
- サブフォームエリアがサブテーブルを表示している場合には、重要な意味を持たないポインタ

例題

アプリケーション全体を通して、レコードを表示する際には、以下の表示方法に従います。フォーム内に変数vsCurrentRecord がある場合、ユーザーが新しいレコードを処理していれば、"New Record" と表示します。ユーザが5200レコードから成るセレクションの56番目のレコードを処理していれば、56/5200と表示します。

そのためには、オブジェクトメソッドを使用して変数vsCurrentRecordを作成します。その後、このオブジェクトメソッドをコピーして、すべてのフォームに貼り付けます。

```
  ` vsCurrentRecord non-enterable variable object method
 Case of
    :(Form event=On Load)
       C_STRING(31;vsCurrentRecord)
       C_POINTER($vpParentTable)
       C_LONGINT($vlRecordNum)
       $vpParentTable:=Current form table
       $vlRecordNum:=Record number($vpParentTable->)
       Case of
          :($vlRecordNum=-3)
             vsCurrentRecord:="New Record"
          :($vlRecordNum=-1)
             vsCurrentRecord:="No Record"
          :($vlRecordNum>=0)
             vsCurrentRecord:=String(Selected record number($vpParentTable->))+" of "+
             String(Records in selection($vpParentTable->))
       End case
 End case
```

DEFAULT TABLE

DEFAULT TABLE (aTable)

引数	型	説明
aTable	テーブル	→ デフォルトとして設定するテーブル

説明

DEFAULT TABLEコマンドは、aTable をカレントプロセスのデフォルトテーブルとして設定します。

DEFAULT TABLEコマンドが実行されるまで、デフォルトテーブルは存在しません。デフォルトテーブルを設定した後で、テーブル引数を省略したコマンドはデフォルトテーブルに対して実行されます。例えば、以下のコマンドを見てください。

```
FORM SET INPUT([table];"form")
```

デフォルトテーブルで[table]を設定した場合に、以下のような同じコマンドの別の記述が可能です。

```
FORM SET INPUT("form")
```

デフォルトテーブルの設定のもう1つの目的は、テーブルに特定されないステートメントを作成することです。これによって、同じステートメントで異なるテーブルを操作することができます。また、テーブルへのポインタを使用して、テーブルに特定されないコードを作成することもできます。この手法に関する詳細は、Table nameコマンドの説明を参照してください。
フィールドを参照する場合テーブル名を省略することはできません。例えば、以下のように記述します:

```
[My Table]My Field:="A string" `正しい記述
```

以下のように記述することはできません。

```
DEFAULT TABLE([My Table])
My Field:="A string" `誤った記述
```

これは、単にデフォルトテーブルが設定されるだけです。しかし、フォームメソッド、オブジェクトメソッドにおいてそれに属するテーブルのフィールドを参照する場合は、テーブル名を省略することができます。

4Dでは、すべてのテーブルは"開かれて" おり、使用する準備ができています。しかし**DEFAULT TABLE**はテーブルを"開いたり" 、カレントテーブルを設定、あるいは入出力のためにテーブルを準備することはありません。**DEFAULT TABLE**はプログラミングの労力の節約とステートメントを読みやすくするための便宜を図るだけです。

Tip: DEFAULT TABLEの使用やテーブル名の省略により、ステートメントを読みやすくすることができるかもしれません。しかし多くのプログラマーは、このコマンドが実際の価値よりも多くの問題と混乱の原因となるとみなしています。特にテーブルフォームとプロジェクトフォームで同じ名前のフォームが存在する場合、**DEFAULT TABLE**が使用されていると、例えばDIALOGコマンドなどでテーブルフォームのほうが使用されてしまいます。

例題

以下の例は、最初に**DEFAULT TABLE**コマンドを使用しないステートメントを示しています。この後で**DEFAULT TABLE**コマンドを使用した同じステートメントを示します。このステートメントは、新しいレコードをデータベースに追加するのによく使用されるループです。FORM SET INPUTコマンドとADD RECORDコマンドは、1番目の引数としてテーブルを必要とします。

```
FORM SET INPUT([Customers];"Add Recs")
Repeat
   ADD RECORD([Customers])
Until(OK=0)
```

デフォルトテーブルの指定により、以下のメソッドが導かれます。

```
DEFAULT TABLE([Customers])
FORM SET INPUT("Add Recs")
Repeat
   ADD RECORD
Until(OK=0)
```

NO DEFAULT TABLE

NO DEFAULT TABLE
このコマンドは引数を必要としません

説明

NO DEFAULT TABLEコマンドを使用して、DEFAULT TABLEコマンドの動作を無効にします。このコマンドを実行した後、プロセスに対して定義されているデフォルトテーブルはありません。
事前にDEFAULT TABLEコマンドが呼び出されていないと、このコマンドはその機能を発揮しません。

このコマンドはプロジェクトフォーム (テーブルにリンクされていないフォーム) の使用と関連しています。
フォーム (ユーザーフォームを除く) に関係するコマンドのほとんどは、任意の引数aTableを最初の引数として受け入れます。例えばFORM GET PARAMETER、Open form window、DIALOGコマンドなどです。プロジェクトフォームとテーブルフォームが同じ名前を持つことができるので、この引数を用いて使用するフォームを決定します。テーブルフォームを使用したい時は引数aTable を渡し、プロジェクトフォームを使用したい時はこの引数を省略します。
[Table1] テーブルに対して、"TheForm" という名前のプロジェクトフォームとそれと同じ名前のテーブルフォームを持つデータベースでは、次のようになります。

```
DIALOG([Table1];"TheForm") `4Dはテーブルフォームを使用
DIALOG("TheForm") `4Dはプロジェクトフォームを使用
```

しかし、同じ名前を持つプロジェクトフォームとテーブルフォームがデータベースにある場合、DEFAULT TABLEコマンドが実行されるとその原理は無効になります。実際この場合に、引数aTableが渡されていなくても、4Dはテーブルフォームをデフォルトで使用します。プロジェクトフォームを確実に使用するには、NO DEFAULT TABLEコマンドを用いなければなりません。

例題

[Table1] テーブルに対して、"TheForm" という名前のプロジェクトフォームとそれと同じ名前のテーブルフォームを持つデータベースでは、次のようになります。

```
DEFAULT TABLE([Table1])
DIALOG("TheForm") `4Dはテーブルフォームを使用
NO DEFAULT TABLE
DIALOG("TheForm") `4Dはプロジェクトフォームを使用
```

デザインオブジェクトアクセス

デザインオブジェクトアクセスコマンド
Current method path
FORM GET NAMES Updated 14.0
METHOD Get attribute
METHOD GET CODE Updated 14.0
METHOD GET COMMENTS
METHOD GET FOLDERS
METHOD GET MODIFICATION DATE
METHOD GET NAMES
METHOD Get path
METHOD GET PATHS
METHOD GET PATHS FORM
METHOD OPEN PATH
METHOD RESOLVE PATH
METHOD SET ACCESS MODE
METHOD SET ATTRIBUTE Updated 14.0
METHOD SET CODE Updated 14.0
METHOD SET COMMENTS

デザインオブジェクトアクセスコマンド

4Dではプログラムを使用してアプリケーションのメソッドコンテンツにアクセスできます。この機能を使用すれば、アプリケーションのコードコントロールツール、特にバージョンコントロールシステム (VCS) への統合が容易になります。またコードのドキュメント化やカスタマイズされたエクスプローラー、ディスクファイルへの定期的なコードバックアップを実装できるようになります。

以下の原則が適用されます:

- 4Dアプリケーション内の各メソッドやフォームはパス名形式で固有のアドレスを持ちます。例えばtable_1のトリガーメソッドは"[trigger]/table_1"で見つけることができます。各オブジェクトパス名はアプリケーション内でユニークです。
  注: パス名がユニークであることを保証するために、異なるフォームページ上であったとしても同じ名前のオブジェクトを作ることはできません。4D v13より前から変換されたデータベースでこの状況が発生している場合、MSCはこのような重複した名前を検出します。
- METHOD GET NAMESやMETHOD GET PATHSのような、このテーマのコマンドを使用して4Dアプリケーション内のオブジェクトにアクセスできます。
- このテーマのコマンドのほとんどはインタープリターモードとコンパイルモード両方で動作します。しかしプロパティを変更したり、メソッドから実行可能なコンテンツにアクセスするコマンドはインタープリターモードでのみ利用可能です (後述の表参照)。
- このテーマのコマンドはすべて4Dローカルモードおよびリモートモードで使用できます。
  しかし特定のコマンドはコンパイルモードで使用することができません。このテーマのコマンドの目的はカスタマイズされた開発サポートツールを作成することにあります。実行中のデータベースの機能を動的に変更するために使用してはなりません。例えばカレントユーザーのアクセス権に基づきMETHOD SET ATTRIBUTEコマンドでメソッド属性を変更するというようなことはできません。
- このテーマのコマンドがコンポーネントから呼ばれた場合、デフォルトでその呼び出しはコンポーネントオブジェクトにアクセスします。コンポーネントからホストデータベースオブジェクトにアクセスする場合は最後に引数に * を渡します。なお (METHOD SET ATTRIBUTEのような) メソッド属性を変更するコマンドを使用する場合、必ずこのシンタックスを使用しなければならない点に留意してください。コンポーネントメソッドは常に読み込みのみモードです。

コンパイルモードでの利用

このテーマのコマンドについて、コンパイルモードで利用可能かどうかを示した表です:

コマンド	**コンパイルモードで利用可**
Current method path	O
FORM GET NAMES	O
METHOD SET ATTRIBUTE	X (*)
METHOD SET CODE	X (*)
METHOD SET COMMENTS	X (*)
METHOD SET ACCESS MODE	O
METHOD Get attribute	O
METHOD Get path	O
METHOD GET PATHS	O
METHOD GET PATHS FORM	O
METHOD GET CODE	X (*)
METHOD GET COMMENTS	O
METHOD GET MODIFICATION DATE	O
METHOD GET FOLDERS	O
METHOD GET NAMES	O
METHOD OPEN PATH	X (*)
METHOD RESOLVE PATH	O

(*) コンパイルモードでこれらのコマンドを使用するとエラー -9762 "このコマンドをコンパイル済みデータベースで実行できません。" が生成されます。

パス名の作成

デフォルトで4Dはディスクファイルを作成することはしません。しかしオブジェクトのために作成されたパス名はOSのファイル管理システムと互換性があります。読み込みや書き出しメソッドを作成し、パス名を使用してディスクに直接ファイルを生成できます。

":"のような使用不可文字はエンコードされます。以下はエンコードが行われる文字のリストです:

文字	**エンコーディング**
"	%22
*	%2A
/	%2F
:	%3A
<	%3C
>	%3E
?	%3F
|	%7C
\	%5C
%	%25

例:
Form?1 は *Form%3F1* にエンコードされます
Button/1 は *Button%2F1* にエンコードされます。

Current method path

Current method path -> 戻り値

引数	型		説明
戻り値	テキスト	↩	実行中のメソッドの内部的な完全パス名

説明

Current method pathコマンドは実行中のデータベースメソッド、トリガー、プロジェクトメソッド、フォームメソッド、またはオブジェクトメソッドの内部的なパス名を返します。

Current method nameと異なり、返される値はデータベース中でユニークである点に留意してください。

注: 4Dマクロコマンドのコンテキストでは、*<method_path>*タグが実行中のメソッドのフルパス名で置き換えられます。

FORM GET NAMES

FORM GET NAMES ({aTable ;} arrNames {; filter}{; marker}{; *})

引数	型		説明
aTable	テーブル	→	テーブル参照
arrNames	テキスト配列	←	フォーム名の配列
filter	テキスト	→	名前のフィルター
marker	倍長整数	→ ←	返す最古のカウンター 最新のカウンター
*	演算子	→	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

FORM GET NAMESコマンドはアプリケーション中のフォーム名をarrNames配列に返します。

aTable引数を渡すと、コマンドはそのテーブルに属するテーブルフォームの名前を返します。この引数を省略するとデータベースプロジェクトフォーム名が返されます。

filter引数に比較文字列を渡すことでフォームのリストを制限できます。この場合、フィルターにマッチする名前を持つフォームだけが返されます。"@"をワイルドカードとして使用することができます。空の文字列を渡した場合、filter引数は無視されます。

任意の marker 引数を使うことによってもフォームのリストを制限することができます。使用した場合、arrNames に返されるフォームを、ある時点以降に変更されたものだけに制限することが出来ます。バージョンコントロールシステムの一部として、この引数は最後のバックアップ以降に変更されたフォームのみをアップデートすることを可能にします。

原理は以下の通りです。4Dは内部にフォーム変更のカウンターを維持しています。フォームが作成または保存されるたびに、このカウンターは数字が増加され、カレントの値は作成または保存されたフォームの内部カウンターに保存されます。 marker 引数を渡すと、コマンドは arrNames の中に、内部カウンターの値がmarker の値以上のフォームのみを返します。さらに、marker の中に変数を渡すと、コマンドはカウンターの新しい値を返します。つまり、最大の値をこの変数の中に返すという事です。この値を保存することによって、次の **FORM GET NAMES** の呼び出しの際に新しく作成または変更されたフォームのみを取得することができます。

コマンドがコンポーネント内で実行された場合、デフォルトではコンポーネントのプロジェクトフォーム名が返されます。* 引数を渡すとホストデータベースのフォーム名を受け取ることができます。

注: ゴミ箱内のフォームは返されません。

例題

典型的な使用例:

```
  // データベース中すべてのプロジェクトフォームを取得
 FORM GET NAMES(arr_Names)

  // [Employees]テーブルのフォームを取得
 FORM GET NAMES([Employees] ;arr_Names)

  // [Employees]テーブル中"input_"で始まるフォームを取得
 FORM GET NAMES([Employees] ;arr_Names;"input_@")

  // データベース中で"dialogue_"から始まるプロジェクトフォームを取得
 FORM GET NAMES(arr_Names;"dialogue_@")

  // コンポーネントからホストのテーブルフォームを取得
  // テーブル名が不明なためポインターを使用
 FORM GET NAMES(tablePtr->;arr_Names;*)
```

METHOD Get attribute

METHOD Get attribute (path ; attribType {; *}) -> 戻り値

引数	型		説明
path	テキスト	⇒	プロジェクトメソッドのパス
attribType	倍長整数	⇒	取得する属性タイプ
*	演算子	⇒	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)
戻り値	ブール	⇐	True: 属性が選択されている、False: 選択されていない

説明

METHOD Get attributeコマンドはpath引数で指定されたプロジェクトメソッドのattribType属性値を返します。このコマンドはプロジェクトメソッドに対してのみ動作します。無効なパスを渡すとエラーが生成されます。

attribType引数には取得する属性のタイプを指定する値を渡します。Design Object Accessテーマの以下の定数を使用できます:

定数	型	値	コメント
Attribute Executed on server	倍長整数	8	"サーバー上で実行"オプションに対応
Attribute Invisible	倍長整数	1	"隠す"オプションに対応
Attribute Published SOAP	倍長整数	3	"Webサービスとして提供"オプションに対応
Attribute Published SQL	倍長整数	7	"SQL利用可"オプションに対応
Attribute Published Web	倍長整数	2	"4DタグおよびURL (4DACTION...) で利用可"オプションに対応
Attribute Published WSDL	倍長整数	4	"WSDLで公開する"オプションに対応
Attribute Shared	倍長整数	5	"コンポーネントとホストデータベースで共有する"オプションに対応

コマンドがコンポーネントから実行された場合、デフォルトでコンポーネントメソッドに適用されます。* 引数を渡すとホストデータベースのメソッドに適用されます。

コマンドは属性が選択されている場合Trueを、選択されていない場合Falseを返します。

METHOD GET CODE

METHOD GET CODE (path ; code {; *})

引数	型		説明
path	テキスト, テキスト配列	→	メソッドパスを格納したテキストまたはテキスト配列
code	テキスト, テキスト配列	←	指定したメソッドのコード
*	演算子	→	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD GET CODEコマンドはpath引数で指定したメソッドの内容をcodeに返します。このコマンドはデータベースメソッド、トリガー、プロジェクトメソッド、フォームメソッド、そしてオブジェクトメソッド等すべてのタイプのメソッドコードを返すことができます。

テキスト配列またはテキスト変数に基づく2つのシンタックスを使用できます:

```
C_TEXT(tVpath) // テキスト変数
C_TEXT(tVcode)
METHOD GET CODE(tVpath;tVcode) // 1つのメソッドのコード

ARRAY TEXT(arrPaths;0) // テキスト配列
ARRAY TEXT(arrCodes;0)
METHOD GET CODE(arrPaths;arrCodes) // 複数メソッドのコード
```

2つのシンタックスを混合して使用することはできません。

無効なパス名を渡すと、code 引数は空のままエラーが生成されます。

このコマンドからcodeに返されるテキストは以下のとおりです:

- 4Dコマンド名はフランス語バージョンを除き英語で記述されます。
- 読み込み時にメタデータが含まれていたコードには、先頭に行が追加されます。例えば:

  ```
  // %attributes = {"lang":"fr","invisible":true,"folder":"Web3"}
  ```

  読み込み時にはこの行は取り除かれ、指定された属性のみが考慮されます。"lang"属性は書き出し言語を設定し、異なる言語への読み込みを防止する目的で使用されます(この場合、エラーが生成されます)。「フォルダ」属性にはメソッドの親フォルダの名前が入ります。メソッドが親フォルダを持っていない場合は表示されません。

コマンドがコンポーネントから実行されると、デフォルトでコンポーネントメソッドに適用されます。* 引数を渡すとホストデータベースにアクセスします。

例題

METHOD SET CODEコマンドの例題参照。

METHOD GET COMMENTS

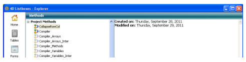
METHOD GET COMMENTS (path ; comments {; *})

引数	型		説明
path	テキスト, テキスト配列	⇨	メソッドパスを格納したテキストまたはテキスト配列
comments	テキスト, テキスト配列	⇦	指定したメソッドのコメント
*	演算子	⇨	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD GET COMMENTSコマンドはpath引数で指定したメソッドのコメントをcomments引数に返します。

このコマンドを使用して取得することのできるコメントは、4Dエクスプローラーのコメント欄で定義されたものです ([METHOD GET CODE](#)コマンドを使用して取得できるコード内のコメントではありません):



4D Listboxes - Explorer
Methods
Home
Tables
Forms
Project Methods
CollapseRowCol
Compiler
Compiler_Arrays
Compiler_Arrays_Inter
Compiler_Methods
Compiler_Variables
Compiler_Variables_Inter

Created on: Thursday, September 28, 2011
Modified on: Thursday, September 29, 2011

このコメントはトリガー、プロジェクトメソッド、そしてフォームメソッドに対して作成できます。またスタイルを付けることができます。

注: フォームとフォームメソッドは同じコメントを共有します。

テキスト配列またはテキスト変数に基づく2つのシンタックスを使用できます:

```
C_TEXT(tVpath) // テキスト変数
C_TEXT(tVcomments)
METHOD GET COMMENTS(tVpath;tVcomments) // 1つのメソッドのコメント

ARRAY TEXT(arrPaths;0) // テキスト配列
ARRAY TEXT(arrComments;0)
METHOD GET COMMENTS(arrPaths;arrComments) // 複数メソッドのコメント
```

2つのシンタックスを混合して使用することはできません。

コマンドがコンポーネントから実行されると、デフォルトでコンポーネントメソッドに適用されます。* 引数を渡すとホストデータベースのメソッドにアクセスします。

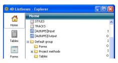
METHOD GET FOLDERS

METHOD GET FOLDERS (arrNames {; filter}{; *})

引数	型		説明
arrNames	テキスト配列	←	ホームページのフォルダー名配列
filter	テキスト	→	名前フィルター
*	演算子	→	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD GET FOLDERSコマンドは4Dエクスプローラーのホームページに作成されたフォルダー名をarrNames配列に返します。


4D Listboxes - Explorer
Home
Tables
Forms
Home
STYLES
TRACKS
[ALBUMS]Input
[ALBUMS]Output
Default group
Forms
Project methods
Tables
1
0
1
0
1
0

フォルダー名はユニークでなければならないため、この配列に階層は返されません。

filter引数に比較文字列を渡して、このフォルダーリストを絞り込むことができます。この場合コマンドは名前がフィルターにマッチするフォルダーのみを返します。"@"をワイルドカードとして使用することができます。空の文字列を渡すとfilter引数は無視されます。

このコマンドがコンポーネントから呼ばれると、デフォルトでコンポーネントのフォルダー名が返されます。* 引数を渡すと配列にはホストデータベースのフォルダーが返されます。

METHOD GET MODIFICATION DATE

METHOD GET MODIFICATION DATE (path ; modDate ; modTime {; *})

引数	型		説明
path	テキスト, テキスト配列	⇒	メソッドパスを格納したテキストまたはテキスト配列
modDate	日付, 日付配列	⇐	メソッド更新日
modTime	時間, 倍長整数配列	⇐	メソッド更新時刻
*	演算子	⇒	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD GET MODIFICATION DATEコマンドはpath引数で指定されたメソッドの更新日と時刻をそれぞれmodDateとmodTime引数に返します。

配列または変数に基づく2タイプのシンタックスを使用できます:

```
C_TEXT(tVpath) // 変数
C_DATE(vDate)
C_TIME(vTime)
METHOD GET MODIFICATION DATE(tVpath;vDate;vTime) // 1つのメソッドの日付と時刻

ARRAY TEXT(arrPaths;0) // 配列
ARRAY DATE(arrDates;0)
ARRAY LONGINT(arrTimes;0)
METHOD GET MODIFICATION DATE(arrPaths;arrDates;arrTimes) // 複数メソッドの日付と時刻
```

この2つのシンタックスを混合して使用することはできません。

コマンドがコンポーネントから実行されると、デフォルトでコンポーネントメソッドに適用されます。* 引数を渡すとホストデータベースのメソッドにアクセスします。

METHOD GET NAMES

METHOD GET NAMES (arrNames {; filter}{; *})

引数	型		説明
arrNames	テキスト配列	⇐	プロジェクトメソッド名配列
filter	テキスト	⇒	名前フィルター
*	演算子	⇒	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD GET NAMESコマンドはアプリケーション中のプロジェクトメソッドの名前をarrNames配列に返します。

デフォルトではすべてのメソッドがリストされます。filter引数に比較文字列を渡して、このリストを絞り込むことができます。この場合コマンドは名前がフィルターにマッチするメソッドのみを返します。"@"をワイルドカードとして使用することができます。空の文字列を渡すとfilter引数は無視されます。

このコマンドがコンポーネントから呼ばれると、デフォルトでコンポーネントプロジェクトメソッドの名前が返されます。* 引数を渡すと配列にはホストデータベースのプロジェクトメソッドが返されます。

注: ゴミ箱内のメソッドは含まれません。

例題

典型的な利用例:

```
  // データベースのすべてのプロジェクト名をリストする
 METHOD GET NAMES(t_Names)

  // 特定の文字から始まるメソッドのみをリストする
 METHOD GET NAMES(t_Names;"web_@")

  // ホストデータベース内で、特定の文字から始まるメソッドのみをリストする
 METHOD GET NAMES(t_Names;"web_@";*)
```

METHOD Get path

METHOD Get path (methodType {; aTable}{; objectName{; formObjectName}}{; *}) -> 戻り値

引数	型		説明
methodType	倍長整数	→	オブジェクトタイプセレクター
aTable	テーブル	→	テーブル参照
objectName	テキスト	→	フォームまたはデータベースメソッド名
formObjectName	テキスト	←	フォームオブジェクト名
*	演算子	→	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)
戻り値	テキスト	↩	オブジェクトのフルパス

説明

METHOD Get pathコマンドはメソッドの完全な内部パス名を返します。

methodType引数にはパスを取得したいメソッドのタイプを渡します。Design Object Accessテーマの以下の定数を使用できます:

定数	型	値	コメント
Path Database method	倍長整数	2	指定したデータベースメソッド名。以下のメソッドのリスト: *[databaseMethod]/onStartup* *[databaseMethod]/onExit* *[databaseMethod]/onDrop* *[databaseMethod]/onBackupStartup* *[databaseMethod]/onBackupShutdown* *[databaseMethod]/onWebConnection* *[databaseMethod]/onWebAuthentication* *[databaseMethod]/onWebSessionSuspend* *[databaseMethod]/onServerStartup* *[databaseMethod]/onServerShutdown* *[databaseMethod]/onServerOpenConnexion* *[databaseMethod]/onServerCloseConnection* *[databaseMethod]/onSystemEvent* *[databaseMethod]/onSqlAuthentication*
Path Project form	倍長整数	4	プロジェクトフォームメソッドとすべてのフォームオブジェクトメソッドのパス。例: *[projectForm]/myForm/{formMethod}* *[projectForm]/myForm/button1* *[projectForm]/myForm/my%2list* *[projectForm]/myForm/button1*
Path Project method	倍長整数	1	メソッド名。 例: *MyProjectMethod*
Path Table form	倍長整数	16	テーブルフォームメソッドとすべてのフォームオブジェクトメソッド。例: *[tableForm]/table_1/Form1/{formMethod}* *[tableForm]/table_1/Form1/button1* *[tableForm]/table_1/Form1/my%2list* *[tableForm]/table_2/Form1/my%2list*
Path Trigger	倍長整数	8	データベーストリガーのパス。例: *[trigger]/table_1* *[trigger]/table_2*

aTable, objectName および formObjectName 引数にはメソッドパス名を取得したいオブジェクトのタイプに応じて値を渡します:

Type of object	aTable	objectName	formObjectName
Path Project form		O	O (オプション)
Path Table form	O	O	O (オプション)
Path Database method		O	
Path Project method		O	
Path Trigger	O		

オブジェクトが見つからない場合 (メソッドタイプが未知か無効、テーブルが見つからない等)、エラーが生成されます。

コマンドがコンポーネントから実行された場合、デフォルトでコンポーネントメソッドのパスが返されます。* 引数を渡すと配列にはホストデータベースのメソッドパス名が返されます。

例題

```
  // "On Startup"データベースメソッドのパス名を取得:
 $path:=METHOD Get path(Path Database method;"onStartup")

  // [Employees]テーブルのトリガーのパス名を取得:
 $path:=METHOD Get path(Path Trigger;[Employees])

  // [Employees]テーブルの"input"フォームの"OK"オブジェクトメソッドのパス名を取得:
 $path:=METHOD Get path(Path Table form;[Employees];"input";"OK")
```

METHOD GET PATHS

METHOD GET PATHS ({folderName ;} methodType ; arrPaths {; stamp}{; *})

引数	型		説明
folderName	テキスト	→	ホームページのフォルダー名
methodType	倍長整数	→	取得するメソッドタイプセレクター
arrPaths	テキスト配列	←	メソッドパスおよび名前の配列
stamp	倍長整数変数	→ ←	スタンプの最小値 新しい現在値
*	演算子	→	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD GET PATHSコマンドはアプリケーションのうち、methodType引数で指定したタイプであるメソッドの内部的なパス名と名前をarrPaths配列に返します。

メソッドが4Dエクスプローラーのホームページでフォルダーを使用して階層化されていれば、オプションのfolderName引数にフォルダー名を渡すことができます。この場合arrPaths配列にはこのフォルダーに含まれるメソッドのみのパスが返されます。

注: folderNameに"@"文字は使用できません。

methodType 引数にはarrPaths配列にパスを取得したいメソッドのタイプを渡します。Design Object Accessテーマの以下の定数を個別にあるいは加算して使用できます:

定数	型	値	コメント
Path All objects	倍長整数	31	データベースのすべてのメソッドのパス
Path Database method	倍長整数	2	指定したデータベースメソッド名。以下のメソッドのリスト: *[databaseMethod]/onStartup* *[databaseMethod]/onExit* *[databaseMethod]/onDrop* *[databaseMethod]/onBackupStartup* *[databaseMethod]/onBackupShutdown* *[databaseMethod]/onWebConnection* *[databaseMethod]/onWebAuthentication* *[databaseMethod]/onWebSessionSuspend* *[databaseMethod]/onServerStartup* *[databaseMethod]/onServerShutdown* *[databaseMethod]/onServerOpenConnexion* *[databaseMethod]/onServerCloseConnection* *[databaseMethod]/onSystemEvent* *[databaseMethod]/onSqlAuthentication*
Path Project form	倍長整数	4	プロジェクトフォームメソッドとすべてのフォームオブジェクトメソッドのパス。例: *[projectForm]/myForm/{formMethod}* *[projectForm]/myForm/button1* *[projectForm]/myForm/my%2list* *[projectForm]/myForm/button1*
Path Project method	倍長整数	1	メソッド名。 例: *MyProjectMethod*
Path Table form	倍長整数	16	テーブルフォームメソッドとすべてのフォームオブジェクトメソッド。例: *[tableForm]/table_1/Form1/{formMethod}* *[tableForm]/table_1/Form1/button1* *[tableForm]/table_1/Form1/my%2list* *[tableForm]/table_2/Form1/my%2list*
Path Trigger	倍長整数	8	データベーストリガーのパス。例: *[trigger]/table_1* *[trigger]/table_2*

stamp引数を使用すれば特定の時点以降に更新されたメソッドのパスだけを取得できます。バージョンコントロールシステムの一部として、最新のバックアップ以降に更新されたメソッドだけをアップデートするようにできます。
動作の概要: 4Dはメソッド更新カウンターを保持します。メソッドが作成され再び保存されるたびに、このカウンターは増分され、その現在値がメソッドの内部的なスタンプに格納されます。
stamp引数を渡すと、コマンドはこの引数に渡された値と同じかそれより大きなスタンプ値を持つメソッドのみを返します。さらにコマンドはstamp引数に更新カウンターの新しい値を返します。この値を保持すれば、次回このコマンドを呼び出す際にこの値を渡すことができ、前回のコマンド実行以降に作成あるいは更新されたメソッドのみを取得できます。

メソッドがコンポーネントから実行された場合、デフォルトでコンポーネントメソッドのパスが返されます。* 引数を渡すと配列にはホストデータベースのメソッドパスが返されます。

例題 1

"web"フォルダー内のプロジェクトメソッドを取得する:

```
METHOD GET PATHS("web", Path Project method;arrPaths)
```

例題 2

データベースメソッドとトリガーを取得する:

```
METHOD GET PATHS(Path Trigger+Path Database method;arrPaths)
```

例題 3

最新のバックアップ以降に更新されたプロジェクトメソッドを取得する:

```
  // 前回のスタンプ値を参照
$stamp :=Max([Backups]cur_stamp)
METHOD GET PATHS(Path Project method;arrPaths;$stamp)
  // 最新のスタンプ値を保存
CREATE RECORD([Backups])
[Backups]cur_stamp :=$stamp
SAVE RECORD([Backups])
```

例題 4

METHOD SET CODEコマンドの例題参照。

METHOD GET PATHS FORM

METHOD GET PATHS FORM ({aTable ;} arrPaths {; filter}{; stamp}{; *})

引数	型		説明
aTable	テーブル	→	テーブル参照
arrPaths	テキスト配列	←	メソッドパスと名前の配列
filter	テキスト	→	名前フィルター
stamp	倍長整数変数	→ ←	スタンプの最小値 新しい現在値
*	演算子	→	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD GET PATHS FORMコマンドはすべてのフォームオブジェクトとフォームメソッドの内部的なパス名と名前をarrPaths配列に返します。
フォームメソッドには{formMethod}とラベルが付けられます。

コードを含むオブジェクトのみがリストされます。例えば標準アクションのみが割り当てられたボタンは返されません。

aTable引数を渡すと、コマンドはこのテーブルのテーブルフォームのオブジェクトを返します。この引数を省略するとコマンドはデータベースプロジェクトフォームのオブジェクトを返します。

filter引数に比較文字列を渡すことでリストを制限できます。この場合フィルターにマッチするフォームだけが返されます。"@"をワイルドカード文字として使用できます。空の文字列を渡すとfilter引数は無視されます。

stamp引数を使用すれば特定の時点以降に更新されたメソッドのパスだけを取得できます。バージョンコントロールシステムの一部として、最新のバックアップ以降に更新されたメソッドだけをアップデートするようにできます。
動作の概要: 4Dはメソッド更新カウンターを保持します。メソッドが作成され再び保存されるたびに、このカウンターは増分され、その現在値がメソッドの内部的なスタンプに格納されます。
stamp引数を渡すと、コマンドはこの引数に渡された値と同じかそれより大きなスタンプ値を持つメソッドのみを返します。さらにコマンドはstamp引数に更新カウンターの新しい値を返します。この値を保持すれば、次回このコマンドを呼び出す際にこの値を渡すことができ、前回のコマンド実行以降に作成あるいは更新されたメソッドのみを取得できます。

メソッドがコンポーネントから実行された場合、デフォルトでコンポーネントメソッドのパスが返されます。* 引数を渡すと配列にはホストデータベースのメソッドパスが返されます。

注: コマンドは継承フォームやサブフォームのオブジェクトを返しません。

例題 1

[Employees]テーブルの"input"フォームのすべてのオブジェクトをリストします。テーブルフォームメソッド (そしてプロジェクトフォームメソッド) はフォームに属するオブジェクトとして処理される点に留意してください:

```
METHOD GET PATHS FORM([Employees];arrPaths;"input")
 // arrPaths配列の内容例
 // [tableForm]/input/{formMethod} -> フォームメソッド
 // [tableForm]/input/bOK -> オブジェクトメソッド
 // [tableForm]/input/bCancel -> オブジェクトメソッド
```

例題 2

"dial"プロジェクトフォームのオブジェクトをリスト:

```
METHOD GET PATHS FORM(arrPaths;"dial")
```

例題 3

コンポーネントから[Employees]テーブル中"input"で始まるフォームのすべてのオブジェクトをリスト:

```
METHOD GET PATHS FORM(([Employees];arrPaths;"input@";*)
```

METHOD OPEN PATH

METHOD OPEN PATH (path {; *})

引数	型		説明
path	テキスト	→	開くメソッドのパス
*	演算子	→	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD OPEN PATHコマンドは内部パス名がpath引数であるメソッドを4Dメソッドエディターで開きます。

このコマンドですべてのタイプ (オブジェクト、フォーム、トリガー、プロジェクト、データベース) のメソッドを開くことができますが、メソッドは既に存在していなければなりません。path 引数が存在しないメソッドを指している場合、エラー -9801 "メソッドを開けません" が生成されます。

コマンドをコンポーネントから実行することもできますが、この場合 (コンポーネントコードには書き込みアクセスができないため) * 引数を渡さなければなりません。この状況で * 引数を省略するとエラー-9763が生成されます。

METHOD RESOLVE PATH

METHOD RESOLVE PATH (path ; methodType ; ptrTable ; objectName ; formObjectName {; *})

引数	型		説明
path	テキスト	→	解決するパス
methodType	倍長整数	←	オブジェクトタイプセレクター
ptrTable	ポインター	←	テーブル参照
objectName	テキスト	←	フォームまたはデータベースメソッド名
formObjectName	テキスト	←	フォームオブジェクト名
*	演算子	→	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD RESOLVE PATHコマンドはpath引数に渡された内部パス名を解決し、methodType、ptrTable、objectName、そして formObjectName引数にそれぞれ情報を返します。

methodType 引数はメソッドのタイプを示す値を受け取ります。この値をDesign Object Accessテーマの定数と比較することができます:

定数	型	値	コメント
Path Database method	倍長整数	2	指定したデータベースメソッド名。以下のメソッドのリスト: *[databaseMethod]/onStartup* *[databaseMethod]/onExit* *[databaseMethod]/onDrop* *[databaseMethod]/onBackupStartup* *[databaseMethod]/onBackupShutdown* *[databaseMethod]/onWebConnection* *[databaseMethod]/onWebAuthentication* *[databaseMethod]/onWebSessionSuspend* *[databaseMethod]/onServerStartup* *[databaseMethod]/onServerShutdown* *[databaseMethod]/onServerOpenConnexion* *[databaseMethod]/onServerCloseConnection* *[databaseMethod]/onSystemEvent* *[databaseMethod]/onSqlAuthentication*
Path Project form	倍長整数	4	プロジェクトフォームメソッドとすべてのフォームオブジェクトメソッドのパス。例: *[projectForm]/myForm/{formMethod}* *[projectForm]/myForm/button1* *[projectForm]/myForm/my%2list* *[projectForm]/myForm/button1*
Path Project method	倍長整数	1	メソッド名。 例: *MyProjectMethod*
Path Table form	倍長整数	16	テーブルフォームメソッドとすべてのフォームオブジェクトメソッド。例: *[tableForm]/table_1/Form1/{formMethod}* *[tableForm]/table_1/Form1/button1* *[tableForm]/table_1/Form1/my%2list* *[tableForm]/table_2/Form1/my%2list*
Path Trigger	倍長整数	8	データベーストリガーのパス。例: *[trigger]/table_1* *[trigger]/table_2*

ptrTable 引数は、パスがテーブルフォームメソッドやトリガーを参照する場合、データベーステーブルへのポインターを受け取ります。

objectName 引数は以下のいずれかを受け取ります:

- パスがテーブルフォームまたはプロジェクトフォームを参照する場合、フォーム名
- パスがデータベースメソッドを参照する場合、データベースメソッド名

formObjectName 引数は、パスがオブジェクトメソッドを参照する場合、フォームオブジェクト名を受け取ります。

コマンドがコンポーネントから実行された場合、デフォルトではコンポーネントメソッドに関する情報を返します。* 引数を渡すと、ホストデータベースメソッドに関する情報を返します。

例題 1

データベースメソッドパスの解決:

```
C_LONGINT($methodType)
C_POINTER($tablePtr)
C_TEXT($objectName)
C_TEXT($formObjectName)

METHOD RESOLVE PATH("[databaseMethod]/onStartup";$methodType;$tablePtr;$objectName;$formObjectName)
 // $methodType: 2
 // $tablePtr: Nil ポインター
 // $objectName: "onStartup"
 // $formObjectName: ""
```

例題 2

テーブルフォームのオブジェクトメソッドのパス解決:

```
C_LONGINT($methodType)
C_POINTER($tablePtr)
C_TEXT($objectName)
C_TEXT($formObjectName)

METHOD RESOLVE PATH("[tabletForm]/Table1/output%2A1/myVar%2A1";$methodType;$tablePtr;$objectName;$formObjectName)
 // $methodType: 16
 // $tablePtr: -> [Table1]
 // $objectName: "output*1"
 // $formObjectName: "Btn*1"
```

METHOD SET ACCESS MODE

METHOD SET ACCESS MODE (mode)

引数	型		説明
mode	倍長整数	→	ロックされたオブジェクトのアクセスモード

説明

METHOD SET ACCESS MODEコマンドはすでに他のユーザーやプロセスにより更新のためにロードされたオブジェクトに書き込みアクセスを行おうとした時の振る舞いを設定します。このコマンドのスコープはカレントセッションです。

mode引数にはDesign Object Accessテーマの以下の定数を渡します:

定数	型	値	コメント
On object locked abort	倍長整数	0	オブジェクトのロードが中断された (デフォルト動作)
On object locked confirm	倍長整数	2	再試行するか中断するか選択できるようにするためダイアログを表示します。リモートモードではこのオプションはサポートされません (ロードは中断されます)。
On object locked retry	倍長整数	1	オブジェクトがリリースされるまでオブジェクトのロードを試行します。

METHOD SET ATTRIBUTE

METHOD SET ATTRIBUTE (path ; attribType ; attribValue {; attribType2 ; attribValue2 ; ... ; attribTypeN ; attribValueN}{; *})

引数	型		説明
path	テキスト	→	プロジェクトメソッドのパス
attribType	倍長整数	→	属性タイプ
attribValue	ブール, テキスト	→	True: 属性を選択 False: 属性の選択解除
*	演算子	→	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD SET ATTRIBUTEコマンドはpath引数で指定されたプロジェクトメソッドのattribType属性の値を設定します。このコマンドはプロジェクトメソッドに対してのみ動作します。無効な pathを渡すとエラーが生成されます。

attribType引数には設定を行う属性のタイプを示す値を渡します。Design Object Accessテーマにある、以下の定数を使用できます:

定数	型	値	コメント
Attribute Executed on server	倍長整数	8	"サーバー上で実行"オプションに対応
Attribute folder name	倍長整数	1024	メソッドのためのフォルダ名(「フォルダ」属性)。この定数を渡した場合、フォルダ名を attribValue に渡す必要があります。 • この名前が有効なフォルダに対応する場合、メソッドはその親フォルダに置かれます。 • フォルダが存在しない場合、コマンドは親フォルダの階層は何も変更しません。 • 空の文字列を渡した場合、メソッドはルート階層に置かれます。
Attribute Invisible	倍長整数	1	"隠す"オプションに対応
Attribute Published SOAP	倍長整数	3	"Webサービスとして提供"オプションに対応
Attribute Published SQL	倍長整数	7	"SQL利用可"オプションに対応
Attribute Published Web	倍長整数	2	"4DタグおよびURL (4DACTION...) で利用可"オプションに対応
Attribute Published WSDL	倍長整数	4	"WSDLで公開する"オプションに対応
Attribute Shared	倍長整数	5	"コンポーネントとホストデータベースで共有する"オプションに対応

attribValue 引数には、以下の値を渡すことが出きます。

- 対応するオプションを選択するには**True**を渡します。選択を解除するには**False**を渡します。
- Attribute folder name 定数を attribType に使用した場合は文字列(フォルダ名)を渡します。

一度の呼び出しに複数のattribType;attribValue ペアを渡す事ができます。

このコマンドをコンポーネントから実行することもできますが、この場合 (コンポーネントコードには書き込みアクセスができないため) * 引数を渡さなければなりません。この状況で * 引数を省略するとエラー-9763が生成されます。

このコマンドをコンパイルモードで実行することはできません。このモードで呼び出されると、エラー -9762 が生成されます。

例題 1

"Choose dialog"プロジェクトメソッドの"コンポーネントとホストデータベースで共有"プロパティを選択します:

```
 METHOD SET ATTRIBUTE("Choose dialog";Attribute Shared;True)
```

例題 2

複数の属性/値のペアを設定する場合:

```
 METHOD SET ATTRIBUTE(vPath;Attribute Invisible;vInvisible;Attribute Published Web;v4DAction;Attribute Published
SOAP;vSoap;Attribute Published WSDL;vWSDL;Attribute Shared;vExported;Attribute Published SQL;vSQL;Attribute
Executed on server;vRemote;Attribute folder name;vFolder;*)
```

METHOD SET CODE

METHOD SET CODE (path ; code {; *})

引数	型		説明
path	テキスト, テキスト配列	⇒	メソッドパスを格納したテキストまたはテキスト配列
code	テキスト, テキスト配列	⇒	指定したメソッドのコード
*	演算子	⇒	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD SET CODEコマンドはpath引数で指定したメソッドのコードをcode引数に渡した内容で置き換えます。このコマンドはデータベースメソッド、トリガー、プロジェクトメソッド、フォームメソッド、オブジェクトメソッドなどすべてのタイプのメソッドのコードにアクセスできます。

プロジェクトメソッド: データベースにメソッドが既に存在する場合、内容が置換されます。存在しなければ新しく作成されます。

テキスト配列あるいはテキスト変数を使用する2通りのシンタックスを使用できます:

```
C_TEXT(tVpath) // テキスト変数
C_TEXT(tVcode)
METHOD SET CODE(tVpath;tVcode) // ひとつのメソッドのコード
```

```
ARRAY TEXT(arrPaths;0) // テキスト配列
ARRAY TEXT(arrCodes;0)
METHOD SET CODE(arrPaths;arrCodes) // 複数のメソッドのコード
```

この2つのシンタックスを混合して使用することはできません。

無効なパス名を渡した場合、コマンドはなにも行いません。

メソッドコードの先頭行に有効なメタデータ(JSON表記)が含まれる場合、それらはメソッド属性を指定するために使用され、先頭行はコードには含まれません。メタデータの例は以下の通りです:

```
 // %attributes = {invisible:true,lang:"fr"}
```

注: これらのメタデータはMETHOD GET CODEコマンドで自動的に生成されます。

メタデータの「フォルダ」プロパティに関しては、以下の点に留意して下さい。

- このプロパティが明示されていて有効なフォルダに対応する場合、メソッドはその親フォルダに置かれます。
- このプロパティが明示されていないかフォルダが存在しない場合、コマンドは親フォルダの階層には何の変更も加えません。
- このプロパティが空の文字列を含む場合、メソッドはルート階層に置かれます。

このコマンドをコンポーネントから実行することもできますが、この場合 (コンポーネントコードには書き込みアクセスができないため) * 引数を渡さなければなりません。この状況で * 引数を省略するとエラー-9763が生成されます。

例題

この例題ではアプリケーションのすべてのメソッドを書き出し/読み込みします:

```
$root_t:=Get 4D folder(Database Folder)+"methods"+Folder separator
ARRAY TEXT($fileNames_at;0)
CONFIRM("メソッドを読み込みますか、書き出しますか？";"Import";"Export")

If(OK=1)
   DOCUMENT LIST($root_t;$fileNames_at)
   For($loop_l;1;Size of array($fileNames_at))
      $filename_t:=$fileNames_at{$loop_l}
      DOCUMENT TO BLOB($root_t+$filename_t;$blob_x)
      METHOD SET CODE($filename_t;BLOB to text($blob_x;UTF8 Text without length))
   End for
Else
   If(Test path name($root_t)#Is a folder)
      CREATE FOLDER($root_t;*)
   End if
   METHOD GET PATHS(Path Project method;$fileNames_at)
   METHOD GET CODE($fileNames_at;$code_at)
   For($loop_l;1;Size of array($fileNames_at))
      $filename_t:=$fileNames_at{$loop_l}
      SET BLOB SIZE($blob_x;0)
      TEXT TO BLOB($code_at{$loop_l};$blob_x;UTF8 Text without length)
      BLOB TO DOCUMENT($root_t+$filename_t;$blob_x;*)
   End for
End if
SHOW ON DISK($root_t)
```

METHOD SET COMMENTS

METHOD SET COMMENTS (path ; comments {; *})

引数	型		説明
path	テキスト, テキスト配列	⇒	メソッドパスを格納したテキストまたはテキスト配列
comments	テキスト, テキスト配列	⇒	指定したメソッドに設定するコメント
*	演算子	⇒	指定時 = コンポーネントで実行されたとき、コマンドはホストデータベースに適用される (コンポーネントのコンテキスト以外ではこの引数は無視されます)

説明

METHOD SET COMMENTSコマンドはpath引数で指定したメソッドのコメントをcomments引数の内容で置き換えます。

このコマンドを使用して更新することのできるコメントは、4Dエクスプローラーのコメント欄で定義されたものです (METHOD GET CODEコマンドを使用して取得できるコード内のコメントではありません)。このコメントはトリガー、プロジェクトメソッド、そしてフォームメソッドに対して作成できます。またスタイルを付けることができます。

注: フォームとフォームメソッドは同じコメントを共有します。

テキスト配列またはテキスト変数に基づく2つのシンタックスを使用できます:

```
C_TEXT(tVpath) // テキスト変数
C_TEXT(tVcomments)
METHOD SET COMMENTS(tVpath;tVcomments) // 1つのメソッドのコメント

ARRAY TEXT(arrPaths;0) // テキスト配列
ARRAY TEXT(arrComments;0)
METHOD SET COMMENTS(arrPaths;arrComments) // 複数メソッドのコメント
```

2つのシンタックスを混合して使用することはできません。

無効なパス名を指定するとエラーが生成されます。

コマンドをコンポーネントから実行することもできますが、この場合 (コンポーネントコードには書き込みアクセスができないため) * 引数を渡さなければなりません。この状況で * 引数を省略するとエラー-9763が生成されます。

例題

既存のトリガーコメントに更新日を追加する:

```
METHOD GET COMMENTS("[trigger]/Table1";$comments)
$comments:="Modif:"+String(Current date)+"\r"+$comments
METHOD SET COMMENTS("[trigger]/Table1";$comments)
```

ドラッグ＆ドロップ

- ドラッグ＆ドロップ
- On Dropデータベースメソッド
- DRAG AND DROP PROPERTIES
- Drop position
- SET DRAG ICON New 14.0

ドラッグ & ドロップ

4Dにはフォームやアプリケーションのオブジェクト間で動作する組み込のドラック＆ドロップ機能があります。一つのオブジェクトを同一のウインドウ内、または別のウインドウにドラック＆ドロップすることが出来ます。言い換えれば、同一のプロセスまたは別のプロセスに対してドラック＆ドロップすることが出来ます。

またオブジェクトを4Dフォームと他のアプリケーション間でドラッグ＆ドロップできます。例えばGIFピクチャを4Dのピクチャフィールドにドラッグ＆ドロップできます。またワードプロセッサアプリケーションでテキストを選択し、4Dのテキスト変数にドロップできます。

そして、フォームが最前面になくとも、アプリケーションに直接オブジェクトをドロップできます。On Dropデータベースメソッドを使用して、このケースのドラッグ＆ドロップを管理できます。つまり、例えば、4Dアプリケーションアイコンに4D Writeドキュメントをドロップして開くことができます。

注: まず最初に、ドラッグ&ドロップアクションがある点から別の点までいくつかのデータを"転送させる"ものとします。あとでドラッグ&ドロップがあらゆるデータタイプの操作メタファである点を見ます。

ドラック＆ドロップオブジェクトプロパティ

あるオブジェクトから別のオブジェクトにドラッグ&ドロップを実行するには、プロパティリストウインドウでそのオブジェクト用の**ドラッグ可**プロパティを選択する必要があります。ドラッグ&ドロップ処理では、ドラッグされるオブジェクトが**ソースオブジェクト**になります。

あるオブジェクトをドラッグ&ドロップ処理のドロップ先にするには、プロパティリストウインドウでそのオブジェクトの**ドロップ可**プロパティを選択する必要があります。ドラッグ&ドロップ処理では、データを受け取るオブジェクトが**ドロップ先オブジェクト**になります。

自動ドラッグと自動ドロップ: これら追加のプロパティは、テキストフィールドや変数、コンボボックス、そしてリストボックスで利用できます。自動ドロップオプションはピクチャフィール ドや変数でも利用できます。これらは内容のコピーに基づく自動ドラッグ＆ドロップを有効にするために使用できます (4Dフォームイベントによるドラッグ＆ドロップアクションの管理は行われなくなります)。この節の最後の"自動ドラッグ＆ドロップ"の段落を参照してください。

デフォルトで、新しく作成されたオブジェクトはドラッグもドロップもできません。これらのプロパティを設定するかどうかは開発者に任されています。

入力フォームまたはダイアログのフォームにあるすべてのオブジェクトは、ドラッグまたはドロップの対象にできます。配列の個別の要素(例えば、スクロール可能エリア)や階層リスト項目、リストボックスの行はドラッグ&ドロップができます。逆に、配列の個別の要素や階層リスト項目、リストボックスの行に対してオブジェクトをドラッグ&ドロップすることもできます。ただし、出力フォームの詳細エリアからオブジェクトをドラッグ& ドロップすることはできません。

アプリケーションのフォーム外へのドラッグアンドドロップも、On Dropデータベースメソッドで管理できます。

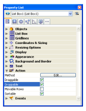
全ての任意のタイプのアクティブオブジェクト(フィールドや変数)をソースおよびドロップ先として使用できるため、ドラッグ&ドロップのユーザインタフェースは簡単に作成できます。例えば、ボタンをドラッグ&ドロップできます。

注:

- "ドラッグ可"に設定されたテキストやボタンをドラッグするには、まず**Alt** (Windows) や **Option** (Mac OS) キーを押します。
- デフォルトで、ピクチャフィールドや変数の場合は、ピクチャとその参照は両方ともドラッグされます。変数やフィールドの参照のみをドラッグしたい場合は、**Alt**ボタン(Windows)または**Option**ボタン(Mac OS)を押す必要があります。
- リストボックスオブジェクトで"ドラック可"と"行の移動可"が同時に選択されている場合、行が移動された場合は"行の移動可"が優先されます。この場合ドラッグはできません。

ドラッグとドロップの両方ができるオブジェクトは、開発者が禁止しない限り、自分自身にもドロップできます。詳細については、以下の説明を参照してください。

以下の図は、選択したオブジェクトに対してプロパティリストウインドウでドラッグ可プロパティとドロップ可プロパティを設定した状態を示しています:



Method
Draggable ☑
Droppable ☑
Movable Rows ☐
Sortable ☑

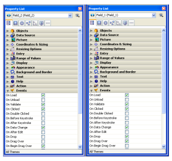
ドラッグ & ドロップのプログラムによる処理

プログラムによるドラッグ & ドロップの管理は、3つのフォームイベントに基づきます: On Begin Drag Over, On Drag Over そして On Drop。

On Begin Drag Overイベントはドラッグの**ソースオブジェクトのコンテキストで生成される**点に注意してください。対してOn Drag OverとOn Dropは**ドロップ先オブジェクトに送信されます**。

アプリケーションがこれらのイベントを処理するためには、プロパティリストで正しく選択されていなければなりません:

プロパティリスト:



Field_2 (Field_2):
On Load ☑
On Unload ☐
On Validate ☑
On Clicked ☑
On Double Clicked ☐
On Before Keystroke ☐
On After Keystroke ☐
On Data Change ☑
On After Edit ☐
On Drop ☐
On Drag Over ☐
On Begin Drag Over ☑

Field_1 (Field_1):
On Load ☑
On Unload ☐
On Validate ☑
On Clicked ☑
On Double Clicked ☐
On Before Keystroke ☐
On After Keystroke ☐
On Data Change ☑
On After Edit ☐
On Drop ☑
On Drag Over ☑
On Begin Drag Over ☐

On Begin Drag Over

On Begin Drag Over フォームイベントは、ドラッグ可能なすべてのフォームオブジェクトで選択できます。このイベントは、オブジェクトが**ドラッグ可**プロパティを持っている場合、すべてのケースで生成されます。

On Drag Over フォームイベントと異なり、On Begin Drag Over はドラッグアクションのソースオブジェクトのコンテキストで呼び出されます。ソースオブジェクトのメソッドあるいはソースオブジェクトのフォームメソッドから呼び出されます。
このイベントはドラッグアクションの高度な管理に使用できます。例えば:

- ペーストボードから (GET PASTEBOARD DATA コマンドを使用して) データと署名を取得する。
- ペーストボードに (APPEND DATA TO PASTEBOARD コマンドを使用して) データと署名を追加する。
- ドラッグの最中に(SET DRAG ICON コマンドを使用して)カスタムのアイコンを表示する。
- ド ラッグされたオブジェクトのメソッドで$0を使用してドラッグを許可/拒否する。ドラッグアクションを受け入れるには、ソースオブジェクトのメソッドは ($0:=0を実行して) 0 (ゼロ) を返さなければなりません。ドラッグアクションを拒否するには、ソースオブジェクトのメソッドは ($0:=-1を実行して) -1 (マイナス1) を返さなければなりません。結果が返されない場合、4Dはドラッグが受け入れられたと判断します。

4Dデータは、イベントが呼び出される前に、ペーストボードに置かれます。例えば、**自動ドラッグ**アクションなしでドラッグした場合、ドラッグされたテキストは、イベントが呼び出される時にはペーストボードにあります。

On Drag Over

On Drag Overイベントは、マウスポインタがオブジェクトの上を移動する時に、繰り返しドロップ先オブジェクトに送られます。このイベントの応答として、開発者は通常、以下のことを行います:

- DRAG AND DROP PROPERTIESコマンドを呼び出して、ソースオブジェクトに関する情報を得ます。
- (現在オブジェクトメソッドが実行されている) ドロップ先オブジェクトとソースオブジェクトの状態やタイプに基づき、ドラッグ&ドロップの**受け付け**または**拒否**を行います。

ドラッグを受け付けるには、ドロップ先のオブジェクトメソッドが ($0:=0を実行して) 0 (ゼロ) を返さなければなりません。ドラッグを拒否するには、オブジェクトメソッドが ($0:=-1を実行して) -1 (マイナス1)を返さなければなりません。$0:=-1と記述します。On Drag Overイベント中、4Dはこのオブジェクトメソッドを関数として扱います。結果が返されない場合には、4Dはドラッグが受け付けられたものと認識します。

ドラッグを受け入れると、ドロップ先オブジェクトがハイライトされます。ドラッグを拒否した場合、ドロップ先オブジェクトはハイライトされません。ドラッグを受け付けることは、ドラッグされたデータがドロップ先オブジェクトに挿入されるという意味ではありません。これは、単にマウスボタンをこの場所 で離したときに、ドラッグされたデータがドロップ先オブジェクトによって受け付けられたことを意味するだけです。

ドロップ可能なオブジェクトに対して開発者がOn Drag Overイベントを処理しない場合には、そのオブジェクトは、ドラッグされたデータの性質やタイプに関係なく、すべてのドラッグ処理に対してハイライトされます。

On Drag Overイベントは、ドラッグ&ドロップ処理の最初の段 階を制御する手段です。ドラッグされたデータがドロップ先オブジェクトと互換性のあるタイプかどうかをテストでき、またドラッグの受け付けや拒否をできる だけでなく、4Dがあなたの判断に基づいてドロップ先オブジェクトをハイライト(または無反応)されるため、この操作が有効であることを操作者にフィード バックすることができます。

On Drag Overイベントはマウスの移動に従って、現在のドロップ先オブジェクトに対して繰り返し送られるため、このイベントのコード処理は短く、短時間で実行されるようにしてください。

警告: 4D v11より、ドラッグ&ドロップが**プロセス間のドラッグ&ドロップ**である場合、つまりソースオブジェクトがドロップ先オブジェクトのプロセスとは異なるプロセス(ウインドウ)にある場合、On Drag Overイベントに対するドロップ先オブジェクトのオブジェクトメソッドは**ドロップ先プロセスのコンテキスト内**で実行されます。ドラッグされた要素の値を得るには、プロセス間通信コマンドを使用しなければなりません。通常このケースでは、On Begin Drag Overイベントとペーストボードテーマのコマンドを使用することが推奨されます。

On Drop

On Dropイベントはマウスポインタがドロップ先オブジェクトに対して離されたときにそのオブジェクトに一度送られます。このイベントはドラッグ&ドロップ処理の第2段階であり、ユーザアクションの応答として処理を実行します。

このイベントは、On Drag Overイベント中にドラッグが受け付けられなかった場合には、オブジェクトに送られません。オブジェクトに対してOn Drag Overイベントを処理し、ドラッグを拒否した場合には、On Dropイベントは発生しません。つまり、On Drag Overイベント中にソースオブジェクトとドロップ先オブジェクト間のデータタイプの互換性をテストして、有効なドロップを受け付けた場合には、On Drop中にデータの再テストをする必要はありません。データがドロップ先オブジェクトに対して適切であることは既にわかっているためです。

4Dのドラッグ&ドロップを実現する上での興味深い点は、必要なことは何でもできるというところです。例えば:

- 階層リストの項目がテキストフィールドに対してドロップされる場合、テキストフィールドの最初、最後、または途中にリスト項目のテキストを挿入できます。
- フォームにゴミ箱のピクチャボタンを配置します。ピクチャには、2つの状態のピクチャが含まれており、一つはごみ箱が空、もう一方はいっぱいであるかを表わすものとします。オブジェクトをゴミ箱ボタンにドロップすることは、(ユーザインタフェースの立場からすると)ごみ箱にドラッグされドロップされたオブジェクトを削除することです。この時、ドラッグ&ドロップはある場所から別の場所にデータを移動しませんが、その代わりにアクションを実行します。
- 顧客リストを表示するフローティングウインドウを作成します。リストは配列をスクロールエリア等として配置します。このリストから別のオブジェクトへドラッグさせ、ドロップした顧客のレコードを表示するということが考えられます。
- など

上記の例からもおわかりのように、4Dのドラッグ&ドロップインタフェースは任意のユーザインタフェースメタファを実現できるフレームワークです。

ドラック＆ドロップコマンド

DRAG AND DROP PROPERTIESコマンドは、以下を返します:

- ドラッグされたオブジェクト(フィールドまたは変数)へのポインタ
- ドラッグされたオブジェクトが配列要素またはリスト項目の場合には、配列要素番号または項目番号。
- ソースプロセスのプロセス番号

Drop positionコマンドは、ドロップ先オブジェクトが配列(スクロール可能エリアなど)、階層リスト、コンボボックスの場合にターゲット要素やリスト項目の項目位置、あるいはリストオブジェクトの場合に列番号を返します。

RESOLVE POINTERやTypeのようなコマンドは、ソースオブジェクトの属性やタイプを調べる際に有効です。

ドラッグされたデータをコピーする目的でドラッグ＆ドラッグ操作が行われた場合、これらのコマンドの機能は関わっているプロセスの数によって変わります:

- ドラッグ＆ドロップが1つのプロセスに限定されている場合、これらのコマンドを使い、適切な動作を実行します(例えばソースオブジェクトをドロップ先オブジェクトに代入するなど)。
- ドラッグ＆ドロップがプロセス間のドラッグ＆ドロップである場合、ドラッグされたデータへのアクセスには注意が必要です。ソースプロセスのデータインスタンスにアクセスしなくてはなりません。ドラッグされたデータが変数の場合、GET PROCESS VARIABLEを使用して正しい値を取得できます。ドラッグされたデータがフィールドの場合、これら2つのプロセスではテーブルのカレントレコードが異なる可能性が高いということに留意してください。正しいレコードにアクセスする必要があります。通常このケースでは、On Begin Drag Overイベントとペーストボードテーマのコマンドを使用することが推奨されます。

ドラッグ&ドロップがデータを移動を目的としたものではなく、独特な処理のためのユーザインタフェースメタファである場合、希望するいかなる処理もプログラミングすることが出来ます。

ペーストボードテーマのコマンド

ドラッグ＆ドロップ操作に、2つの4Dアプリケーション間または4Dアプリケーションとサードパーティーアプリケーション間でのデータやドキュメントの移動が関わる場合、"ペーストボード"テーマのコマンドが必要なツールを提供します。
実 際、これらのコマンドはコピー/ペーストとデータのドラッグ＆ドロップ両方の管理に使用できます。4Dは2つのペーストボードを使用します。1つはデータのコピー (またはカット) に使用される実際のクリップボードで、もう1つはドラッグおよびドロップされるデータのためのものです。これら2つのペーストボードは、同じコマンドを使 用して管理されます。コンテキストによりどちらかにアクセスします。

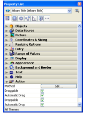
ドラッグ＆ドロップ操作に関する"ペーストボード"テーマのコマンドの詳細は、ペーストボードの管理を参照してください。

自動ドラッグ＆ドロップ

テキストエリア (フィールド, 変数, コンボボックス そしてリストボックス) やピクチャオブジェクトは自動ドラッグ＆ドロップをサポートし、シングルクリックであるエリアから他のエリアにテキストやピクチャを移動またはコピーでき ます。これは同じ4Dエリア、2つの4Dエリア、4Dと他のアプリケーション間で使用できます。

Note: 2つの4Dエリア間の自動ドラッグ＆ドロップの場合データは移動されます。言い換えれば、データはソースエリアから削除されます。データをコピーしたい場合、アクションの間**Ctrl** (Windows) または **Command** (Mac OS) を押します。

自動ドラッグ＆ドロップはフォームオブジェクトごとに、プロパティリストの2つのオプション、**自動ドラッグ**と**自動ドロップ**で個別に設定できます:



Method
Draggable ☑
Automatic Drag ☑
Droppable ☑
Automatic Drop ☑

- **自動ドラッグ** (テキスト型オブジェクトのみ): このオプションがをチェックすると、オブジェクトの自動ドラッグモードが有効になります。このモードでは、On Begin Dragフォームイベントは生成されません。
自動ドラッグが有効のときに標準のドラッグを強制したい場合、アクションの間**Alt** (Windows) または **Option** (Mac OS) キーを押しながら操作します。このオプションはピクチャでは利用できません。
- **自動ドロップ**: このオプションは自動ドロップモードを有効にするために使用します。4Dは可能な限り自動で、オブジェクトにドロップされたテキストやピクチャ型データの挿入を管理します (データはオブジェクトにペーストされます)。この場合On Drag Over と On Drop フォームイベントは生成されません。他方、ドロップ中の On After Edit とオブジェクトがフォーカスを失った時の On Data Change イベントは生成されます。
テキストやピクチャ以外のデータ (他の4Dオブジェクトやファイルなど) や複合データがドロップされた場合、アプリケーションは ドロップ可オプションを参照します。オプションがチェックされていれば、On Drag Over と On Drop フォームイベントが生成されます。そうでなければドロップは拒否されます。これは"外部からのドラッグアンドドロップを拒否"オプションの設定も関連します (以下参照)。

外部からのドラッグアンドドロップを拒否 (互換性)

バージョン11より、4Dはピクチャファイルなど外部のセレクションやオブジェクト、ファイルのドラッグアンドドロップをサポートするようになりました。これはデータベースコードでサポートされなければなりません。
以前のバージョンの4Dから変換されたデータベースでは、既存のコードが対応していなければおかしな動作につながることになります。このため、環境設定の**外部からのドラッグアンドドロップを拒否**オプションでこの機能を無効にできます。このオプションはアプリケーション/互換性のページにあります。変換されたデータベースではデフォルトでこのオプションがチェックされています。

このオプションがチェックされていると、外部オブジェクトの4Dフォームへのドロップは拒否されます。しかし、**自動ドロップ**オプションにより、外部オブジェクトの挿入が依然可能である点に留意してください。この場合アプリケーションがドロップされたテキストまたはピクチャデータを解釈できます。

On Dropデータベースメソッド

On Dropデータベースメソッドはローカルおよびリモートモードの4Dで使用できます。

このデータベースメソッドは、オブジェクトが4Dアプリケーションのフォームやウィンドウの外にドロップされると自動で実行されます。例えば:

- MDIウィンドウの空のエリア (Windows)
- Dockやシステムデスクトップの4Dアイコン (Mac OS)

Mac OSでは、データベースメソッドが呼び出されるためにはOption+Commandキーがドロップの際に押されていなければなりません。

デスクトップで4Dアプリケーションアイコンにドロップが行われると、アプリケーションが4D Volume Desktopにマージされている場合を除き、**On Dropデータベースメソッド**はアプリケーションが既に起動されている場合にのみ呼び出されます。マージされている場合、アプリケーションが起動されていなくてもデータベースメソッドは呼び出されます。これはカスタムのドキュメント署名を定義できることを意味します。

例題

この例題は、フォーム外側にドロップされた4D Sriteドキュメントを開きます:

```
  `On Drop database method
 droppedFile:=Get file from pasteboard(1)
 If(Position(".4W7";droppedFile)=Length(droppedFile)-3)
    externalArea:=Open external window(100;100;500;500;0;droppedFile;"_4D Write")
    WR OPEN DOCUMENT(externalArea;droppedFile)
 End if
```

DRAG AND DROP PROPERTIES

DRAG AND DROP PROPERTIES (srcObject ; srcElement ; srcProcess)

引数	型		説明
srcObject	ポインター	←	ドラッグ＆ドロップソースオブジェクトのポインタ
srcElement	倍長整数	←	ドラッグされた配列要素番号, または ドラッグされたリストボックス行番号, または ドラッグされた階層リスト項目, または ソースオブジェクトが配列でもリストボックスでも 階層リストでもない場合、-1
srcProcess	倍長整数	←	ソースプロセス番号

説明

DRAG AND DROP PROPERTIESコマンドを使用して、On Drag OverイベントやOn Dropイベントが(配列、リストボックス、階層リストなどの)"複合"オブジェクト上で発生した時の、ソースオブジェクトについての情報を取得することができます。

特に、On Drag OverイベントまたはOn Dropイベントが発生するドロップ先オブジェクトのオブジェクトメソッド(またはそこから呼び出されるサブルーチン)の内部から**DRAG AND DROP PROPERTIES**コマンドを使用します。

重要: フォームオブジェクトは、**ドロップ可** プロパティが選択されている場合にドロップされたデータを受け付けます。また、そのオブジェクトメソッドは、On Drag OverやOn Dropを処理するためにアクティブにする必要があります。

コマンドの呼び出し後:

- srcObject引数は、ソースオブジェクト(ドラッグ＆ドロップするオブジェクト)へのポインタです。このオブジェクトは、ドロップ先オブジェクト(On Drag OverイベントまたはOn Dropイ ベントが発生するオブジェクト)または異なるオブジェクトである可能性がある点に注意してください。同一のオブジェクト間でデータのドラッグとドロップを 実行することは、配列や階層リストでは便利です。これは、ユーザが配列またはリストを手作業でソートする簡単な方法です。
- ドラッグ＆ドロップするデータが配列要素(配列要素のドラッグ)である場合、srcElement引数はその配列の要素番号を返します。リストボックスの行をドラッグされた場合は、srcElement引数はこの行番号を返します。階層リストの項目をドラッグされた場合は、srcElement引数は項目位置番号を返します。上記以外の場合、つまりソースオブジェクトが配列、リストボックスおよび階層リストでもない場合、srcElement引数は-1を返します。
- ドラッグ＆ドロップ処理はプロセス間でも発生します。srcProcess引 数は、ソースオブジェクトが属するプロセス番号と同じです。この引数の値をテストすることが重要です。同一プロセス内のドラッグ＆ドロップに応答するのは、単にソースデータからドロップ先オブジェクトにコピーするだけです。一方でプロセス間のドラッグ＆ドロップを扱う場合、GET PROCESS VARIABLEコマンドを使用してソースプロセスのオブジェクトインスタンスからソースデータを取得します。通常ドラッグ＆ドロップはユーザインタフェース内で配列やリストなどのソース変数から、フィールドや変数などのデータ入力エリアに実装されます。

  互換性に関する注意: 4Dバージョン11より、特にインタープロセス間では、ドラッグ＆ドロップを On Begin Drag Over イベントとペーストボードテーマのコマンドを使用して管理することが推奨されます。

ドラッグ＆ドロップイベント以外で**DRAG AND DROP PROPERTIES**コマンドを呼び出すと、引数srcObjectはNILポインタを返し、srcElementは-1を、srcProcessは0を返します。

Tip: 4Dはドラッグ＆ドロップのグラフィカルな部分を自動的に処理します。次に開発者は適切な方法でイベントに応答する必要があります。以下の例で は、イベントに応答してドラッグされたデータをコピーしています。別の方法として洗練されたユーザインタフェースをインプリメントできます。例えば、フローティングウインドウから配列要素をドラッグ＆ドロップすることにより、ドロップ先ウインドウ(ドロップ先オブジェクトが存在するウインドウ) に構造化されたデータ(例: 対象配列要素によって特定されたレコードのいくつかのフィールド)を入れるような処理です。

On Drag Overイベント中にドロップ先オブジェクトがドラッグ＆ドロップ処理を受け付けるかどうかをドラッグオブジェクトのタイプや性質(または他の理由)に従って判断するには、**DRAG AND DROP PROPERTIES**コマンドを使用します。ドラッグ＆ドロップを受け付ける場合には、オブジェクトメソッドは$0:=0を返す必要があります。ドラッグ＆ドロップを受け付けない場合には、オブジェクトメソッドは$0:=-1を返す必要があります。ドラッグ＆ドロップを受け付けたか、拒否したかは、画面に反映されます。つまり、オブジェクトがドラッグ＆ドロップ処理のドロップ先となる場合にはハイライトされ、ドロップ先にならない場合にはハイライトされません。

例題 1

複数のデータベースフォームの中にスクロールエリアがあり、そのスクロールエリアのある部分から別の部分へドラッグ＆ドロップするだけで要素の順 序を手作業で変えたい場合があります。それぞれの状況に応じて特定のコードを書くよりも、これらのスクロールエリアを処理する汎用プロジェクトメソッドを 作成することができます。以下のようなコードを作成できます:

```
 ` Handle self array drag and drop プロジェクトメソッド
 ` Handle self array drag and drop (ポインタ) -> ブール
 ` Handle self array drag and drop ( ->array) -> Is a self array drag and drop
Case of
   :(Form event=On Drag Over)
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
      If($vpSrcObj=$1)
 ` 自身の配列からのドラッグ＆ドロップなら受け付ける
         $0:=0
      Else
         $0:=-1
      End if
   :(Form event=On Drop)
 ` ドラッグ＆ドロップソースオブジェクトの情報を取得
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
 ` ドロップ先要素番号を取得
      $vlDstElem:=Drop position
 ` 自身の要素へのドロップでなければ
      If($vlDstElem #$vlSrcElem)
 ` ドラッグされた要素を配列の要素 0 に格納
         $1->{0}:=$1->{$vlSrcElem}
 ` ドラッグされた要素を削除
         DELETE FROM ARRAY($1->;$vlSrcElem)
 ` ドロップ先がソース要素より後なら
         If($vlDstElem>$vlSrcElem)
 ` ドロップ先要素番号をデクリメント
            $vlDstElem:=$vlDstElem-1
         End if
 ` ドラッグアンドドロップが最後の要素より後に発生していたら
         If($vlDstElem=-1)
 ` ドロップ先を配列の新しい最期の要素にセット
            $vlDstElem:=Size of array($1->)+1
         End if
 ` 新しい要素を挿入
         INSERT IN ARRAY($1->;$vlDstElem)
 ` 配列の要素0に格納した値を代入
         $1->{$vlDstElem}:=$1->{0}
 ` この要素を配列の新しい選択要素にする
         $1->:=$vlDstElem
      End if
End case
```

このプロジェクトメソッドを実装したら、以下のように使用できます:

```
 ` anArray スクロールエリアのオブジェクトメソッド

Case of
 `...
   :(Form event=On Drag Over)
      $0:=Handle self array drag and drop(Self)
   :(Form event=On Drop)
      Handle self array drag and drop(Self)
 ` ...
End case
```

例題 2

複数のデータベースフォームで、さまざまなソースからドラッグアンドドロップを受け付けたい入力可テキストエリアがあります。それぞれに特定のコードを書くのではなく、汎用のメソッドを書くことができます。コードは以下のようになります:

```
 ` Handle dropping to text area プロジェクトメソッド
 ` Handle dropping to text area ( Pointer )
 ` Handle dropping to text area ( -> テキストまたは文字列変数 )

Case of
 ` ドラッグ＆ドロップを受け入れまたは拒否するためにこのイベントを使用する
   :(Form event=On Drag Over)
 ` $0を拒否値に初期化する
      $0:=-1
 ` ソースオブジェクトの情報を取得する
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
 ` この例題では、自身のドラッグアンドドロップを許可しない
      If($vpSrcObj #$1)
 ` ドラッグされたデータのタイプを取得
         $vlSrcType:=Type($vpSrcObj->)
         Case of
            :($vlSrcType=Is Alpha Field)
 ` 文字フィールドは受け入れる
               $0:=0
 ` 変数に値をコピー
               <>vtDraggedData:=$vpSrcObj->
            :($vlSrcType=Is Text)
 ` テキストフィールドまたは変数は受け入れる
               $0:=0
               RESOLVE POINTER($vpSrcObj;$vsVarName;$vlTableNum;$vlFieldNum)
 ` フィールドなら
               If(($vlTableNum>0) & ($vlFieldNum>0))
 ` 変数に値をコピー
                  <>vtDraggedData:=$vpSrcObj->
               End if
            :($vlSrcType=Is String Var)
 ` 文字列変数は受け入れる
               $0:=0
            :(($vlSrcType=String array)|($vlSrcType=Text array))
 ` 文字またはテキスト配列は受け入れる
               $0:=0
            :(($vlSrcType=Is LongInt)|($vlSrcType=Is Real)
               If(Is a list($vpSrcObj->))
 ` 階層リストは受け入れる
                  $0:=0
               End if
         End case
      End if

 ` 実際のドラッグ＆ドロップアクションを行うためこのイベントを使用する
   :(Form event=On Drop)
      $vtDraggedData:=""
 ` ソースオブジェクトの情報を取得する
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
      RESOLVE POINTER($vpSrcObj;$vsVarName;$vlTableNum;$vlFieldNum)
 ` フィールドなら
      If(($vlTableNum>0) & ($vlFieldNum>0))
 ` On Drag Over イベントで設定した変数の値を取得
         $vtDraggedData:=<>vtDraggedData
      Else
 ` ドラッグされた変数の型を取得
         $vlSrcType:=Type($vpSrcObj->)
         Case of
 ` 配列なら
            :(($vlSrcType=String array)|($vlSrcType=Text array))
               If($vlPID #Current process)
 ` ソースプロセスの変数インスタンスから要素を取得
                  GET PROCESS VARIABLE($vlPID;$vpSrcObj->{$vlSrcElem};$vtDraggedData)
               Else
 ` 配列要素をコピー
                  $vtDraggedData:=$vpSrcObj->{$vlSrcElem}
               End if
 ` If it is a list
            :(($vlSrcType=Is Real)|($vlSrcType=Is LongInt))
 ` 他のプロセスのリストなら
               If($vlPID #Current process)
 `他のプロセスのリスト参照を取得
                  GET PROCESS VARIABLE($vlPID;$vpSrcObj->;$vlList)
               Else
                  $vlList:=$vpSrcObj->
               End if
 ` リストが存在すれば
               If(Is a list($vpSrcObj->))
 `位置が取得された項目のテキストを取得
                  GET LIST ITEM($vlList;$vlSrcElem;$vlItemRef;$vsItemText)
                  $vtDraggedData:=$vsItemText
               End if
            Else
 ` 文字列またはテキスト変数
               If($vlPID #Current process)
                  GET PROCESS VARIABLE($vlPID;$vpSrcObj->;$vtDraggedData)
               Else
                  $vtDraggedData:=$vpSrcObj->
               End if
         End case
      End if
 ` 実際ドロップされるものがある場合 (ソースオブジェクトが空の場合がある)
      If($vtDraggedData #"")
 ` テキスト変数の文字長が 32,000 文字を超ないかチェック
         If((Length($1->)+Length($vtDraggedData))<=32000)
            $1->:=$1->+$vtDraggedData
         Else
            BEEP
            ALERT("ドラッグ＆ドロップができません。テキストが長すぎます。")
         End if
      End if

End case
```

このプロジェクトメソッドを実装したら、以下のように使用できます:

```
 ` [anyTable]aTextField オブジェクトメソッド

Case of
 ` ...
   :(Form event=On Drag Over)
      $0:=Handle dropping to text area(Self)

   :(Form event=On Drop)
      Handle dropping to text area(Self)
 ` ...
End case
```

例題 3

リストボックスからドラッグされたデータをテキストエリアにコピーする。

label1のオブジェクトメソッドは以下のとおりです:

```
Case of
   :(Form event=On Drag Over)
      DRAG AND DROP PROPERTIES($source;$arrayrow;$processnum)
      If($source=Get pointer("list box1"))
         $0:=0 `ドロップを受け入れる
      Else
         $0:=-1 `ドロップを拒否する
      End if
   :(Form event=On Drop)
      DRAG AND DROP PROPERTIES($source;$arrayrow;$processnum)
      QUERY([Members];[Members]LastName=arrNames{$arrayrow})
      If(Records in selection([Members])#0)
         label1:=[Members]FirstName+" "+[Members]LastName+Char(Carriage return)+[Members]Address
         +Char(Carriage return)+[Members]City+","+" "+[Members]State
         +" "+[Members]ZipCode
      End if
End case
```

以下のようなアクションが可能になります:

Drop position

Drop position {(columnNumber | pictPosY)} -> 戻り値

引数	型		説明
columnNumber | pictPosY	倍長整数	←	リストボックス列番号 (ドロップ位置が最後の列を超えた場合-1)、または ピクチャーの場合Y座標の位置
戻り値	倍長整数	↩	・数値 (配列/リストボックス) または ・位置 (階層リスト) または ・ドロップ先項目の文字列中の位置 (テキスト/コンボボックス) または ・最後の配列要素やリスト項目を超えてドロップされた場合-1 または ・ピクチャー中のX座標の位置

説明

Drop positionコマンドは、"複合" オブジェクトに対して行われたドロップのドロップ位置を知るために使用します。

特に配列、階層リスト、リストボックス、テキストおよびピクチャフィールドに対して発生したドラッグ&ドロップイベントを処理するのに**Drop position**を使用します。

- ドロップ先オブジェクトが配列の場合、このコマンドは要素番号を返します。
- ドロップ先オブジェクトがリストボックスの場合、このコマンドは行番号を返します。この場合、オプションのcolumnNumber引数には列番号が返されます。。
- ドロップ先オブジェクトが階層リストの場合、この関数は項目位置を返します。
- ドロップ先オブジェクトがテキストフィールドや変数、あるいはコンボボックスの場合、コマンドは文字列中の文字位置を返します。
  いずれの場合も、ソースオブジェクトが最後の配列要素または最後のリスト項目を超えてドロップされた場合、このコマンドは-1を返します。
- ドロップ先オブジェクトがピクチャ型の変数やフィールドの場合、コマンドはクリックの縦位置と、オプションのpictPosY引数にクリックの横位置を返します。返される値はローカルの座標システムに対し相対で、ピクセルで表現されます。

配列やリストボックス、コンボボックス、階層リストに対してドラッグ&ドロップイベント以外で**Drop position**を呼び出すと、コマンドは-1を返します。

重要: フォームオブジェクトに**ドロップ可**プロパティが選択されている場合、ドロップされたデータを受け付けます。また、そのオブジェクトメソッドは、On Drag OverまたはOn Dropあるいはその両方に対して、これらのイベントを処理するためにアクティブにする必要があります。

例題 1

DRAG AND DROP PROPERTIES コマンドの例題を参照。

例題 2

以下の例題では、支払い済み金額は月ごとまた人毎にブレークダウンします。これはスクロールエリアからのドラッグ & ドロップで行われます:

リストボックスには以下のオブジェクトメソッドが記述されています:

```
Case of
   :(Form event=On Drag Over)
      DRAG AND DROP PROPERTIES($source;$arrayrow;$processnum)
      If($source=Get pointer("SA1")) `ソースオブジェクトがスクロールエリアなら
         $0:=0
      Else
         $0:=-1 `ドロップを拒否する
      End if
   :(Form event=On Drop)
      DRAG AND DROP PROPERTIES($source;$arrayrow;$processnum)
      $rownum:=Drop position($colnum)
      If($colnum=1)
         BEEP
      Else
         Case of `Adding of dropped values
            :($colnum=2)
               John{$rownum}:=John{$rownum}+SA1{$arrayrow}
            :($colnum=3)
               Mark{$rownum}:=Mark{$rownum}+SA1{$arrayrow}
            :($colnum=4)
               Peter{$rownum}:=Peter{$rownum}+SA1{$arrayrow}
         End case
         DELETE FROM ARRAY(SA1;$arrayrow) `エリアを更新
      End if
End case
```

SET DRAG ICON

SET DRAG ICON (icon {; horOffset {; vertOffset}})

引数	型		説明
icon	ピクチャー	→	ドラッグ中に表示するアイコン
horOffset	倍長整数	→	カーソルから見てアイコンの左端との水平方向の距離を指定(>0 は左方向に、<0 は右方向に移動)
vertOffset	倍長整数	→	カーソルから見てアイコンの上端との垂直方向の距離を指定(>0 は上方向に、<0 は下方向に移動)

説明

SET DRAG ICONコマンドは、指定したアイコンを、ドラッグ＆ドロップ中のカーソルと関連付けます。

このコマンドはOn Begin Drag Over のフォームイベント中（ドラッグ中）しか呼び出すことができません(Form event コマンドを参照のこと)

icon 引数にはドラッグ中に表示させたいピクチャを渡します。サイズは最大で 256x256 ピクセルです。縦か横のどちらかの大きさが256ピクセルを超えていた場合、画像は自動的にリサイズされます。

horOffset と vertOffset ではオフセットの具合をピクセルで指定できます:

- horOffset では、マウスカーソルから見て icon で指定した画像の左端がどれだけ水平方向にオフセットしているかを指定します。正の値を渡すとアイコンが左側に、負の値を渡すとアイコンが右側にずれていきます。
- vertOffset では、マウスカーソルから見て icon で指定した画像の上端がどれだけ垂直方向にオフセットしているかを指定します。正の値を渡すとアイコンが上方向に、負の値を渡すとアイコンが下方向にずれていきます。

引数省略時にはカーソルがアイコンの中央に位置するようになります。

例題

フォーム内で、ユーザーが行をドラッグ＆ドロップするとラベルを生成することができるようにします。この場合リストボックスのオブジェクトメソッドは以下の様になります:

```
If(Form event=On Begin Drag Over)
   READ PICTURE FILE(Get 4D folder(Current Resources Folder)+"splash.png";vpict)
   CREATE THUMBNAIL(vpict;vpict;48;48)
   SET DRAG ICON(vpict)
End if
```

行をドラッグすると、以下のように画像が表示されます:

画像とカーソルの位置関係をずらすこともできます:

```
SET DRAG ICON(vpict;0;0)
```

トランザクション

- トランザクションを使用する
- CANCEL TRANSACTION
- In transaction
- START TRANSACTION
- Transaction level
- VALIDATE TRANSACTION

トランザクションを使用する

トランザクションは、あるプロセスにおいてデータベースに対して行われる一連の関連したデータ更新です。トランザクションは、そのトランザクションが受け入れられるまで、データベースに恒久的には保存されません。キャンセルされたり、他の外部的な原因でトランザクションが完了できなかった場合には、更新処理の結果は保存されません。

トランザクション処理中に、プロセス内でデータベースのデータに対して行った更新はすべて、一時的なバッファにローカルで保存されます。トランザクションがVALIDATE TRANSACTIONで受け入れられた時点で、更新されたデータが恒久的に保存されます。トランザクションがCANCEL TRANSACTIONでキャンセルされた場合、更新されたデータは保存されません。すべての場合において、カレントセレクションやカレントレコードは、トランザクション管理コマンドでは更新されません。

バージョン11以降、4Dはネストされたトランザクション、つまり幾つかの階層レベルにあるトランザクションをサポートしています。認められているサブトランザクションの数は無限です。Transaction levelコマンドを使用して、コードが実行されている現在のトランザクションレベルを調べることができます。
ネストされたトランザクションを使用するとき、各サブトランザクションの結果は、高レベルトランザクションの受け入れまたはキャンセルにより異なります。高レベルトランザクションが受け入れられると、サブトランザクションの結果が承認されます (受け入れまたはキャンセル) 。一方、高レベルトランザクションがキャンセルされると、それぞれの結果に関係なく、すべてのサブトランザクションがキャンセルされます。

互換性のオプション

ネストされたトランザクションは、4Dの以前のバージョンで構築されたデータベース上で誤動作を招きます。そのため、変 換されたデータベースではデフォルトで無効になっています (トランザクションは単一レベルに制限されています) 。変換されたデータベース上で複数レベルのトランザクションを利用したい場合、アプリケーションの環境設定にあるアプリケーション/互換ページの "**トランザクションのネストを許可する**" オプションをチェックして明確に指示しなければなりません。
こ のオプションは変換されたデータベース上にしか表示されません。これは、デフォルトでチェックされていません。この設定は各データベースに固有のものです。4DのSQLエンジンで実行されたトランザクションには効果がありません。SQLトランザクションはいつもマルチレベルです。

トランザクション例題

この例題は、簡単な請求書システムです。請求書の明細は、[Invoice Lines] テーブルに格納されています。[Invoice Lines]Invoice IDフィールドは[Invoices]Invoice IDフィールドとリレートしています。請求書が追加されると、重複不可のIDがSequence numberコマンドを使用して計算されます。[Invoices] テーブルと [Invoice Lines] テーブルのリレートは1対nの自動リレートになっており、サブフォームにあるリレート値自動代入チェックボックスがチェックされています。

[Invoice Lines] テーブルと [Parts] テーブルのリレートは、マニュアルリレートです。

ユーザは請求書を入力する場合に、以下のような動作を実行しなければなりません。

- [Invoices] テーブルにレコードを追加する
- [Invoice Lines] テーブルにレコードを幾つか追加する
- 請求書にリストされた各部品の [Parts]In Warehouseフィールドを更新する

この例題は、トランザクションの使用が必要になる典型的なシーンの1つです。処理中に必ずこれらのレコードをすべて保存できるということや、または レコードが追加されない場合やレコードが更新されない場合は、トランザクションをキャンセルできるということを確認する必要があります。つまり、リレート されたレコードを保存しなくてはなりません。

トランザクションを使用しない場合には、データベースの理論的なデータの整合性を保証することはできません。例えば、[Parts] テーブルのレコードがロックされていると、[Parts]In Warehouseフィールドに格納されている数量を更新することはできません。したがって、このフィールドは理論上、正しいものではなくなります。つまり、販売した部品の合計と倉庫内に残っている在庫数が、レコードに入力したオリジナルの数量と等しくならなくなります。こういう状況を避けるために、トラ ンザクションを使用します。

トランザクションを使って、データ入力を実行する方法は幾つかあります。

1. START TRANSACTION、VALIDATE TRANSACTIONとCANCEL TRANSACTIONトランザクションコマンドを使用してトランザクションを独自に管理できます。例えば、以下のように記述します。

```
READ WRITE([Invoice Lines])
READ WRITE([Parts])
FORM SET INPUT([Invoices];"Input")
Repeat
   START TRANSACTION
   ADD RECORD([Invoices])
   If(OK=1)
      VALIDATE TRANSACTION
   Else
      CANCEL TRANSACTION
   End if
Until(OK=0)
READ ONLY(*)
```

2. データ入力実行中のレコードロックを減らすには、フォームメソッド内からトランザクションを管理し、必要になった時にだけREAD WRITE状態にしてアクセスすることができます。

サブフォームに [Invoice Lines] リレートテーブルを持つ [Invoices] テーブルの入力フォームを使ってデータ入力を行います。このフォームには動作なしボタン属性を持つbCancelとbOKの2つのボタンがあります。
メソッドは以下のようになります。

```
READ WRITE([Invoice Lines])
READ ONLY([Parts])
FORM SET INPUT([Invoices];"Input")
Repeat
   ADD RECORD([Invoices])
Until(bOK=0)
READ ONLY([Invoice Lines])
```

データ入力中は [Parts] テーブルが読み込み専用の状態になっていることに注意してください。読み/書き状態はデータ入力が有効な場合にのみ利用できます。

[Invoices] 入力フォームから開始されるトランザクションを以下に示します。

```
Case of
   :(Form event=On Load)
      START TRANSACTION
      [Invoices]Invoice ID:=Sequence number([Invoices]Invoice ID)
   Else
      [Invoices]Total Invoice:=Sum([Invoice Lines]Total line)
End case
```

bCancelボタンをクリックすると、トランザクションはもちろんのことデータ入力も取り消す必要があります。
以下にbCancelボタンのオブジェクトメソッドを示します。

```
Case of
   :(Form event=On Clicked)
      CANCEL TRANSACTION
      CANCEL
End case
```

bValidateボタンをクリックすると、トランザクションはもちろんのことデータ入力も受け付ける必要があります。以下にbOKボタンのオブジェクトメソッドを示します。

```
Case of
   :(Form event=On Clicked)
      $NbLines:=Records in selection([Invoice Lines])
      READ WRITE([Parts]) `[Parts] テーブルに対して、読み /書き状態に変更する
      FIRST RECORD([Invoice Lines]) `最初の明細で開始する
      $ValidTrans:=True `すべてOKであると仮定する
      For($Line;1;$NbLines) `各明細に対して
         RELATE ONE([Invoice Lines]Part No)
         OK:=1 `続行したいと仮定する
         While(Locked([Parts]) & (OK=1)) `読み /書き状態でレコードを取得してみる
            CONFIRM("The Part "+[Invoice Lines]Part No+" is in use. Wait?")
            If(OK=1)
               DELAY PROCESS(Current process;60)
               LOAD RECORD([Parts])
            End if
         End while
         If(OK=1)
 `倉庫の数量を更新する
            [Parts]In Warehouse:=[Parts]In Warehouse-[Invoice Lines]Quantity
            SAVE RECORD([Parts]) `レコードを保存する
         Else
            $Line:=$NbLines+1 `ループを抜ける
            $ValidTrans:=False
         End if
         NEXT RECORD([Invoice Lines]) `次の明細へ移動する
      End for
      READ ONLY([Parts]) ` テーブルを読み込み専用状態にする
      If($ValidTrans)
         SAVE RECORD([Invoices]) `送り状レコードを保存する
         VALIDATE TRANSACTION `すべてのデータベースの修正を受け入れる
      Else
         CANCEL TRANSACTION `すべてキャンセルする
      End if
      CANCEL `フォームを抜ける
End case
```

このコードでは、ボタンのクリックに関係なく、CANCEL コマンドを実行します。新しいレコードはACCEPT ボタンを呼び出しても受け入れられず、SAVE RECORDコマンドで受け入れられます。さらに、SAVE RECORDコマンドがVALIDATE TRANSACTIONコマンドの直前に呼び出されている点に注意してください。したがって、[Invoices] テーブルのレコードを保存するということは、実際にはトランザクションの一部であるということです。ACCEPT コマンドを呼び出してレコードを受け入れることもできますが、その場合、[Invoices] レコードが保存される前にトランザクションが受け入れられてしまいます。つまり、レコードはトランザクションの外で保存されてしまいます。

必要に応じ、データベースを独自にカスタマイズすることができます。最後の例題では、[Parts] テーブルのロックレコードの処理をさらに開発することも可能です。

CANCEL TRANSACTION

CANCEL TRANSACTION
このコマンドは引数を必要としません

説明

CANCEL TRANSACTIONは、対応するレベルのSTART TRANSACTIONで開始したカレントプロセスのトランザクションをキャンセルします。**CANCEL TRANSACTION**は、トランザクション中にデータ上で実行された処理をキャンセルします。

注: CANCEL TRANSACTIONはまだ保存されていないカレントレコードに対して行われた変更には影響しません。このコマンドが実行された後も変更されたデータはそのまま表示されます。

In transaction

In transaction -> 戻り値

引数	型	説明
戻り値	ブール	↩ カレントプロセスがトランザクション内にある場合、TRUEを返します。

説明

In transactionコマンドはカレントプロセスがトランザクション内にある場合にTrueを返します。その他の場合はFalseを返します。

例題

複数のレコードに対する操作 (レコード追加、修正、または削除) を実行すると、ロックされたレコードに出くわす可能性があります。このような場合、データの整合性を維持するためには、失敗したときに操作全体をロールバックして、データベースを元の状態に戻せるように、トランザクションの中で捜査を行わなければなりません。

トリガ内から、あるいはサブルーチン (トランザクションの中でも、外でも呼び出せる) から操作を実行する場合には、In transactionを使用して、カレントプロセスのメソッドまたは呼び出し側のメソッドがトランザクションを開始したかどうかをチェックすることができます。トランザクションが開始されていなければ、操作を開始してはいけません。そうでなければ失敗した場合に、操作をロールバックすることができなくなってしまいます。

START TRANSACTION

START TRANSACTION
このコマンドは引数を必要としません

説明

START TRANSACTION は、カレントプロセスでトランザクションを開始します。トランザクションが受け入れられるまたはキャンセルされるまでは、トランザクション内にデータベース上で変更されたすべてのデータは一時的に保存されます。

4Dのバージョン11以降、複数のトランザクション(サブトランザクション) をネストすることができます。それぞれのトランザクションまたはサブトランザクションは、最終的にはキャンセルまたは認証されていなければなりません。主要なトランザクションがキャンセルされると、結果に関係なく、すべてのサブトランザクションはキャンセルされますので注意してください。

Transaction level

Transaction level -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	現在のトランザクションレベル (トランザクションが 開始されていない場合は0)

説明

Transaction levelコマンドはプロセスの現在のトランザクションレベルを返します。4Dランゲージ経由またはSQL経由でトランザクションが開始されたかに関わらず、このコマンドはカレントプロセスのすべてのトランザクションを考慮に入れます。

VALIDATE TRANSACTION

VALIDATE TRANSACTION
このコマンドは引数を必要としません

説明

VALIDATE TRANSACTIONは、カレントプロセス中、START TRANSACTIONで開始した対応するレベルのトランザクションを受け入れます。VALIDATE TRANSACTIONは、トランザクション中に行われたデータベースへの更新を保存します。

4Dのバージョン11以降、複数のトランザクション(サブトランザクション) をネストすることができます。主要なトランザクションがキャンセルされると、このコマンドで個別に認証されていても、すべてのサブトランザクションはキャンセルされます。

システム変数およびセット

トランザクションが正しく認証されると、システム変数OKに1が代入されます。その他の場合は0が代入されます。

トリガ

- トリガ
- Trigger event
- Trigger level
- TRIGGER PROPERTIES

トリガ

トリガーはテーブルに付属するメソッドであり、テーブルのプロパティです。トリガーを呼び出す必要はありません。テーブルレコードを操作 (追加、削除、修正) するたびに4Dのデータベースエンジンが自動的に呼び出します。まず簡単なトリガーを記述し、後からより洗練されたものにすることができます。

トリガーを使用すれば、データベースのレコードに対して "不正な" 操作が行われるのを防ぐことができます。偶発的なデータの紛失や改ざんを防ぎ、テーブル上での操作を制限することのできる非常に強力なツールです。例えば請求書システムにおいては、請求書の送付先である顧客を指定せずに誰かが請求書を追加するのを防止することができます。

トリガのアクティブ化と作成

デザインモードでテーブルを作成したときには、デフォルトでテーブルにトリガーがありません。

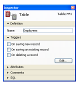
テーブルのトリガーを使用するには、以下を実行する必要があります。

- トリガーをアクティブにし、4Dに対してトリガーをいつ起動すべきか知らせる。
- トリガー用のコードを記述する。

まだメソッドとして記述されていないトリガーをアクティブにする、あるいはトリガーをアクティブにしないでメソッドに記述しても、テーブルに対して実行される操作に影響を与えることはありません。

1. トリガをアクティブにする

テーブルのトリガーをアクティブにするには、ストラクチャーのインスペクターウィンドウでテーブルの**トリガー** オプション (データベースイベント) を選択しなければなりません。



既存レコード保存時

このオプションを選択すると、テーブルのレコードが修正されるたびに、トリガーが起動します。
以下の場合にトリガーが起動します。

- データ入力時にレコードを修正する (デザインモード、MODIFY RECORDコマンド またはMissingRefコマンド等を使用) 。
- SAVE RECORDを使用して既存レコードを保存する。
- 既存レコードを保存するコマンドを使用する (ARRAY TO SELECTION、APPLY TO SELECTIONなど) 。
- SAVE RECORD コマンドを呼び出すプラグインを使用する。

注: 最適化のため、ユーザーがレコードを保存したりSAVE RECORDコマンドでレコードが保存されたりする際、レコードのフィールドが全く変更されていなければ、トリガーは呼び出されません。トリガーを強制的に呼び出したいときは、フィールドに同じ値を代入します:

```
[thetable]thefield:=[thetable]thefield
```

レコード削除時

このオプションを選択すると、テーブルのレコードが削除されるたびに、トリガーが起動します。
以下の場合にトリガが起動します。

- レコードを削除する (デザインモード、DELETE RECORDコマンド、DELETE SELECTIONコマンドまたはSQLのMissingRefコマンドを使用する) 。
- リレートの削除制御オプションによって、リレート先レコードの削除を引き起こす何らかの操作を実行する。
- DELETE RECORDコマンドを呼び出すプラグインを使用する。

注: APPLY TO SELECTIONコマンドはトリガーを呼び出しません。

新規レコード保存時

このオプションを選択すると、レコードがテーブルに追加されるたびに、トリガーが起動します。

以下の場合にトリガーが起動します。

- データ入力時にレコードを追加する(デザインモード、ADD RECORDコマンド または SQLのMissingRefコマンド等を使用) 。
- CREATE RECORD や SAVE RECORDを使用してレコードを作成し、保存する。トリガーはSAVE RECORDを呼び出したときに起動します。レコードを作成したときではありません。
- レコードを読み込む (デザインモード、または読み込みコマンドを使用して) 。
- 新規レコードを作成または保存するコマンドを使用する (ARRAY TO SELECTION、SAVE RELATED ONEなど) 。
- CREATE RECORD や SAVE RECORDコマンドを呼び出すプラグインを使用する。

2. トリガを作成する

テーブルのトリガーを作成するには、**エクスプローラー**を使用するか、ストラクチャーエディターのインスペクターウィンドウにある**編集...** ボタンをクリックするか、Alt (Windowsの場合) または Option (Mac OSの場合) キーを押して、ストラクチャーエディターのテーブルタイトルをダブルクリックしてください。詳細については、4D Design Referenceを参照してください。

データベースイベント

トリガーは、前述の3つの**データベースイベント**のいずれかに対して起動することができます。トリガー内でTrigger event関数を呼び出すことによって、どのイベントが発生しているかを検出します。この関数はデータベースイベントを示す数値を返します。

一般的には、Trigger eventから返される結果に関して、**Case of** ストラクチャーを用いて、トリガーを記述します。LAST SUBRECORDテーマの定数を使用できます。

```
 // トリガー用の[anyTable] テーブル
C_LONGINT($0)
C_LONGINT($event_l)
$0:=0 // データベースリクエストが許可されると仮定する
$event_l:=Trigger event // データベースイベントを取得
Case of
   :($event_l=On Saving New Record Event)
 // 新規に作成されたレコードの保存のために適切な動作 (アクション) を実行する
   :($event_l=On Saving Existing Record Event)
 // 既存のレコードの保存のために適切な動作を実行する
   :($event_l=On Deleting Record Event)
 // レコードの削除のために適切な動作を実行する
End case
```

トリガと関数

トリガーには、2つの目的があります。

- レコードが保存、削除される前に、レコードに対して動作 (アクション) を実行する。
- データベース操作を許可または拒絶する。

1. 動作を実行する

[Documents] テーブルにレコードが保存 (追加または修正) されるたびに、作成時を示すタイムスタンプと最新の修正時を示すタイムスタンプでレコードを "マーク" したいとします。この場合、以下のようなトリガーを記述できます。

```
 `トリガー用の [Documents] テーブル
C_LONGINT($event_l)
$event_l:=Trigger event
Case of
   :($event_l=On Saving New Record Event)
      [Documents]Creation Stamp:=Time stamp
      [Documents]Modification Stamp:=Time stamp
   :($event_l=On Saving Existing Record Event)
      [Documents]Modification Stamp:=Time stamp
End case
```

注: この例題で使用している Time stamp 関数は、固定日付が任意に選択された時点から経過数秒を返す小さなプロジェクトメソッドです。

いったんこのトリガーを記述してアクティブにすると、ユーザーがどのような方法で [Documents] テーブルにレコードを追加または修正しても (データ入力、読み込み、プロジェクトメソッド、4Dプラグイン) 、レコードが最終的にディスクに書き込まれる前に、トリガーによって、[Documents]Creation Stamp フィールドと [Documents]Modification Stamp フィールドに自動的に日付が割り当てられます。

注: この例の詳細についてはGET DOCUMENT PROPERTIESコマンドの例を参照してください。

2. データベース操作を許可または拒絶する

データベース操作を許可または拒絶するために、トリガーは、戻り値 $0 に**トリガーエラーコード**を返さなければなりません。

例題 1

[Employees] テーブルの場合を取り上げてみましょう。データ入力時に、[Employees]Social Security number フィールドで規則を強制します。確認ボタンをクリックする際に、ボタンのオブジェクトメソッドを使用してそのフィールドをチェックします。

```
 ` bAcceptボタンのオブジェクトメソッド
If(Good SS number([Employees]SS number))
   ACCEPT
Else
   BEEP
   ALERT("Enter a Social Security number then click OK again.")
End if
```

フィールド値が有効な場合、データ入力を受け入れます。フィールド値が無効な場合、警告を表示して、データ入力の状態になります。

[Employees] レコードをプログラムで作成した場合、以下のコードはプログラムとしては正当ですが、前述のオブジェクトメソッドで強制した規則に違反します。

```
 `プロジェクトメソッドから抽出する
 ` ...
CREATE RECORD([Employees])
[Employees]Name:="DOE"
SAVE RECORD([Employees]) ` <-- DB規則の違反! 保険証番号は保存されない!
 ` ...
```

[Employees] テーブルのトリガーを使用して、データベースのすべてのレベルで[Employees]SS number の規則を強制することができます。トリガーは以下のようになります。

```
 `[Employees] のトリガー
$0:=0
$dbEvent:=Trigger event
Case of
   :(($dbEvent=On Saving New Record Event)|($dbEvent=On Saving Existing Record Event))
      If(Not(Good SS number([Employees]SS number)))
         $0:=-15050
      Else
 ` ...
      End if
 ` ...
End case
```

いったんこのトリガーを記述しアクティブにすると、SAVE RECORD ([Employees]) 行はデータベースエンジンエラー-15050を生成し、そのレコードは保存されません。

同様に4Dプラグインが無効な保険証番号で[Employees] レコードを保存しようとしても、トリガーは同じエラーを生成しレコードは保存されません。

トリガーを使用すれば、誰も (ユーザー、データベース設計者、プラグイン) 保険証番号の規則を故意にまたは偶発的に違反できないことが保証されます。

テーブルのトリガーが無くても、レコードを保存または削除しようとしているときに、データベースエンジンエラーが生じる場合があるので注意してください。例えば、重複不可属性を持つインデックスフィールドで重複する値を持つレコードを保存しようとすると、エラー-9998が返されます。

したがって、エラーを返すトリガーは、新しいデータベースエンジンエラーをアプリケーションへ追加します。

- 4Dは "通常" エラー、すなわち重複不可のインデックス、リレーショナルデータのコントロールなどを管理します。
- トリガーを使用して、開発者はアプリケーションに固有のカスタムエラーを管理できます。

重要: エラーコード値は任意のものを返すことができます。ただし、4Dデータベースエンジンによって既に確保されているエラーコードは使用できません。-32000 から -15000 の間のエラーコードを使用することを強く勧めます。-15000を超えるエラーコードは、データベースエンジン用に予約されています。

プロセスレベルでは、データベースエンジンエラーと同じ方法で、トリガーエラーを処理します。

- 4Dに標準のエラーダイアログボックスを表示させ、その後メソッドが停止します。
- ON ERR CALLでインストールしたエラー処理メソッドを使用して、適切な方法でエラーから回復します。

注:

- データ入力時に、レコードを受け入れまたは削除しようとしているときにトリガーエラーが返されると、エラーは重複不可なインデックスエラーのように処理されます。エラーダイアログが表示され、データ入力状態になります。デザインモード (アプリケーションモードでなく) でデータベースを使用する場合でも、トリガーを使用することのメリットが得られます。
- レコードのセレクションで動作しているコマンドのフレームワーク内のトリガーによってエラーが生成されると (DELETE SELECTIONのような) 、コマンドの実行は即座に停止し、セレクションは必ずしも完全に処理されません。この場合は、デベロッパーによる適切な処理が必要となります。例えばセレクションを一時的に保存したり、トリガーの実行前にエラーを取り除くなどの処理が必要です。

トリガーがエラーを返さないからといって ($0:=0) 、データベース操作が成功したという意味ではありません。重複不可なインデックス違反が生じる場合があります。操作がレコードの更新である場合、レコードがロックされたり I/O エラーが生じたりすることがあります。トリガーの実行後にチェックが終了します。ただしプロセスを実行する高レベルにおいては、データベースエンジンまたはトリガーによって返されるエラーは同じものです。トリガーエラーはデータベースエンジンエラーです。

トリガと4Dアーキテクチャ

トリガーはデータベースエンジンレベルで実行されます。以下の図にその様子をまとめています。

データベースエンジンが実際に配置されているマシンでトリガーは実行されます。これはシングルユーザー版の4Dでは明白です。4D Serverではクライアントマシンではなく、サーバーマシン (トリガーを起動させるプロセスの "対の" プロセスで) 上で動作しているプロセス内でトリガーは実行されます。

トリガーが起動される場合、トリガーはデータベース操作を実行しようとするプロセスのコンテキスト内で実行されます。トリガーの実行を引き起こすこのプロセスは**起動プロセス**と呼ばれます。
コンテキストに含まれるエレメントは、データベースが4Dのローカルモードで実行されたか、または4D Serverで実行されたかにより異なります。

- 4Dのローカルモードでは、トリガーは起動プロセスのカレントセレクション、カレントレコードテーブルの読み/書き状態、およびレコードロック操作を用いて動作します。
- 4D Serverでは、起動クライアントプロセスのデータベースのコンテキストのみが保持されます (ロックされたレコードやトランザクションの状態など) 。また4D Serverはトリガーのテーブルのカレントレコードが正確に配置されていることのみを保証します。コンテキストの他の要素 (例えばカレントセレクション) はトリガープロセスのものです。

4D環境の他のデータベースオブジェクトや言語オブジェクトは注意して使用してください。これは、トリガーが起動プロセスのマシンとは別のマシン上で実行される可能性があるためです。4D Serverがこれに当てはまります。

- **インタープロセス変数**: トリガーは、トリガーが実行されるマシンのインタープロセス変数へアクセスします。4D Serverでは、トリガーは起動プロセスのマシンとは別のマシンへアクセスすることがあります。
- **プロセス変数**: 各トリガーは独自のプロセス変数テーブルを持っています。トリガーは、トリガーを起動する元となったプロセスのプロセス変数にアクセスすることはできません。
- **ローカル変数**: トリガー内でローカル変数を使用できます。その有効範囲はトリガーの実行中です。ローカル変数はトリガーの実行のたびに作成され、削除されます。
- **セマフォー**: トリガー、(トリガーが実行されるマシン上の) ローカルセマフォーだけでなく、グローバルセマフォーもテスト、または設定できます。ただしトリガーは即座に実行されなければならないため、トリガー内からセマフォーをテストまたは設定する場合には、十分な注意が必要です。
- **セットと命名セレクション**: トリガー内からセットまたは命名セレクションを使用する場合、トリガーが実行されるマシン上で作業することになります。クライアント/サーバーモードでは、クライアントマシン上で作成されるプロセスセットとプロセス命名セレクションは、トリガー内で可視です。
- **ユーザーインターフェース**: トリガー内でユーザーインターフェースエレメントを使用しないでください (警告、メッセージ、ダイアログボックスを使用しない) 。したがって、トリガーのトレースはデバッグウィンドウに限定する必要があります。クライアント/サーバーでは、トリガーは4D Server上で実行されることを覚えておいてください。サーバーマシン上で警告メッセージを表示しても、クライアント上のユーザーの助けにはなりません。起動プロセスにユーザインターフェースの処理も行わせるようにしてください。

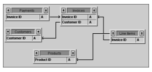
トリガとトランザクション

トランザクションは起動プロセスレベルで処理されなければなりません。トリガーレベルでトランザクションを管理してはいけません。一つのトリガーを実行している間に、複数のレコード (下記の例を参照) を追加、修正、削除する必要がある場合、最初にトリガー内からIn transactionコマンド を使用して、起動プロセスが現在トランザクション内にあるかどうかテストしなければなりません。そうでない場合には、トリガーがロックされたレコードに出くわす可能性があります。そのため、起動プロセスがトランザクション内に無い場合は、レコードに対する操作を開始しないでください。起動プロセスに、実行しようとしているデータベース操作はトランザクション内で実行されなければならないことを知らせるためにエラーを$0 に返すだけにしてください。そうしないとロックされたレコードに出くわした場合、起動プロセスにはトリガーの動作をロールバックする方法がなくなります。

注: トリガーとトランザクションを統合した操作を最適化するため、4DではVALIDATE TRANSACTIONを実行した後、トリガーは呼び出されません。これにより、トリガーの実行を2度繰り返すことを防ぎます。

トリガのカスケード

以下の例のようなストラクチャーがあるとします。



注: 上記のテーブルは簡略化されています。実際には、テーブルにはここに示したよりも多くのフィールドがあります。

データベースがある請求書の削除を "許可" するとしましょう。そのような操作がトリガーレベルでどのように処理されるか検討してみます (プロセスレベルで削除を実行することも可能です) 。

リレートに関するデータの整合性を維持するには、請求書の削除において、[Invoices] のトリガー内で実行される以下の動作が必要となります。

- [Customer] レコードにおいて、送り状の金額分だけ総売上フィールドの額を減らす。
- 送り状に関連したすべての [Line Items] レコードを削除する。
- これはまた、[Line Items] トリガーが、削除された明細品目に関連した [Products] レコードの売り上げ数量フィールドの数量を減らすことを意味する。
- 削除された送り状に関連するすべての [Payments] レコードを削除する。

最初に、[Invoices] のトリガーは、起動プロセスがトランザクション内にある場合に限り、これらの動作を実行しなければなりません。そのため、ロックされたレコードに出くわした場合にロールバックが可能になります。

次に、[Line Items] のトリガーは、[Invoices] のトリガーと**カスケード**しています。明細品目の削除は [Invoices] のトリガー内からDELETE SELECTIONを呼び出した結果であるため、[Line Items] のトリガーは [Invoices] のトリガーの実行の "範囲内で" 実行されます。

この例題にあるすべてのテーブルは、すべてのデータベースイベント対してアクティブなトリガーを持っているとします。トリガーのカスケードは以下のようになります。

- 起動プロセスが請求書を削除するため [Invoices] トリガーが起動される。
  - [Invoices] トリガーが総売上フィールドを更新するため、[Customers] トリガーが起動される。
  - [Invoices] トリガーが明細品目を削除するため (繰り返し) 、[Line Items] トリガーが起動される。
    - [Line Items] トリガーが売上数量フィールドを更新するため、[Products] トリガーが起動される。
  - [Invoices] トリガーが支払を削除するため (繰り返し) 、[Payments] トリガーが起動される。

このカスケードの関係においては、[Invoices] のトリガーはレベル1で、[Customers]、[Line Items]、と [Payments] のトリガーはレベル2で、そして [Products] のトリガーはレベル3で実行されていると言えます。

トリガー内からTrigger levelコマンドを使用して、トリガーが実行されるレベルを検出します。更にTRIGGER PROPERTIESコマンドを使用して、他のレベルに関する情報を入手することができます。

例えば、[Products] レコードがプロセスレベルで削除されている場合、[Products] のトリガーは、レベル3ではなく、レベル1で実行されます。

Trigger levelとTRIGGER PROPERTIESを使用すれば、動作の原因を検出できます。前述の例では、請求書がプロセスレベルで削除されています。[Customers] レコードをプロセスレベルで削除すると、[Customers] のトリガーは、その顧客に関連するすべての請求書を削除しようとします。これにより、前述の例と同じように、[Invoices] のトリガーが起動されることになりますが、起動される理由は異なります。[Invoices] トリガー内から、そのトリガーがレベル1で実行されたか、レベル2で実行されたかを、検出することができます。トリガーがレベル2で実行された場合には、次に、それが [Customers] レコードが削除されたためであるかどうかをチェックできます。そうであれば、総売上フィールドの更新にわずらわされる必要はありません。

トリガ内での通し (シーケンス) 番号の使用

On Saving New Record Event データベースイベントを処理する際に、テーブルのレコードに対してユニークなID番号を維持するために、Sequence numberコマンドを呼び出すことができます。

例題 2

```
 // [Invoices]テーブルのトリガー
Case of
   :(Trigger event=On Saving New Record Event)
 // ...
      [Invoices]Invoice ID Number:=Sequence number([Invoices])
 // ...
End case
```

Trigger event

Trigger event -> 戻り値

引数	型	説明
戻り値	倍長整数	0: トリガー実行サイクル外 1: 新規レコード保存時 2: 既存レコード保存時 3: レコード削除時

説明

Trigger eventコマンドがトリガー内で呼び出されると、データベースイベントのタイプ、つまりそのトリガーが起動された理由を示す数値を返します。

Trigger Eventsテーマに以下のような定義済み定数があります。

定数	型	値
On Deleting Record Event	倍長整数	3
On Saving Existing Record Event	倍長整数	2
On Saving New Record Event	倍長整数	1

トリガー内で複数のレコードに対してデータベース操作を実行すると、トリガーの正確な実行を妨げる条件 (通常レコードのロック) に遭遇します。この状況の例としては、[Invoices] テーブルにレコードを追加しているときに、[Products] テーブルにある複数のレコードを更新する場合が挙げられます。この時点で、必ずデータベースの操作を停止して、データベースのエラーを返さなければなりません。そうすることにより、起動プロセスがそのデータベースリクエストを実行できないことを理解します。起動プロセスはトリガーによって実行された不完全なデータベース操作を、トランザクション中にキャンセルできなければなりません。このような状況が発生した場合、トリガー内から何らかの操作を試みた前からそのプロセスがトランザクション内にあることを確かめる必要があります。これを実行するにはIn transactionコマンドを使用します。

トリガーのカスケードを呼び出す場合、4Dには使用可能なメモリの容量以外に制限はありません。トリガーの実行を最適化するために、データベースイベントだけでなく、トリガーがカスケードされて起動される際の呼び出しのレベルに基づいて、トリガーのコードを記述することもできます。例えば[Invoices] テーブルに対する削除データベースイベントの中で、[Invoices] レコードの削除が、削除された [Customers] レコードに関連した**すべて**の送り状の削除にともなうものである場合には、[Customers] 総売上げフィールドの更新をスキップすることもできます。これを実行するにはTrigger levelとTRIGGER PROPERTIESコマンドを使用します。

例題

Trigger eventコマンドを使用して、以下のようにトリガを作成します。

```
 `[anyTable] 用のトリガー
C_LONGINT($0)
$0:=0 `データベースリクエストが許可されると仮定する
Case of
   :(Trigger event=On Saving New Record Event)
 `新規に作成されたレコードの保存のために適切な動作を実行する
   :(Trigger event=On Saving Existing Record Event)
 `既存のレコードの保存のために適切な動作を実行する
   :(Trigger event=On Deleting Record Event)
 `レコードの削除のために適切な動作を実行する
End case
```

Trigger level

Trigger level -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	トリガの実行レベル (トリガの実行サイクル外であれば0)

説明

Trigger levelコマンドはトリガの実行レベルを返します。

実行レベルについての詳細は、トリガのカスケードの節にあるトリガのカスケードを参照してください。

TRIGGER PROPERTIES

TRIGGER PROPERTIES (triggerLevel ; dbEvent ; tableNum ; recordNum)

引数	型		説明
triggerLevel	倍長整数	→	トリガ実行サイクルレベル
dbEvent	倍長整数	←	データベースイベント
tableNum	倍長整数	←	影響を受けるテーブル番号
recordNum	倍長整数	←	影響を受けるレコード番号

説明

TRIGGER PROPERTIESコマンドは、triggerLevelに渡すトリガーの実行レベルに関する情報を返します。トリガー実行レベルのカスケードに基づいて異なる動作を実行するには、**TRIGGER PROPERTIES**とTrigger levelを組み合わせて使用します。詳細についてはトリガにあるトリガーのカスケードを参照してください。

存在しないトリガー実行レベルを渡すと、コマンドはすべての引数に0を返します。

トリガー実行レベルのデータベースイベントの種類が、引数 dbEvent に返されます。Trigger Eventsテーマに以下のような定義済み定数があります。

定数	型	値
On Deleting Record Event	倍長整数	3
On Saving Existing Record Event	倍長整数	2
On Saving New Record Event	倍長整数	1

トリガー実行レベルのデータベースイベントに関係するレコードのテーブル番号とレコード番号が、引数 tableNum と recordNum に返されます。

バックアップ

- On Backup Shutdownデータベースメソッド
- On Backup Startupデータベースメソッド
- BACKUP
- CHECK LOG FILE
- GET BACKUP INFORMATION
- GET RESTORE INFORMATION
- INTEGRATE LOG FILE
- INTEGRATE MIRROR LOG FILE New 14.0
- Log File
- New log file
- RESTORE
- SELECT LOG FILE

On Backup Shutdownデータベースメソッド

On Backup Shutdownデータベースメソッドは、データベースのバックアップが終了するたびに呼び出されます。バックアップが終了する理由には、コピーの終了、 ユーザによる中断、そしてエラーがあります。
これはすべての4D環境: 4D (すべてのモード), 4D Server、4D Volume Desktopが統合されたコンパイル済みアプリケーションに当てはまります。

On Backup Shutdownデータベースメソッドを使用すると、バックアップが正常に実行されたかどうかを確認できます。バックアップが完了すると、このメソッド内の$1 引数にはバックアップのステータスを示す値が返されます:

- バックアップが正常に終了すると、$1には0が代入されます。
- バックアップがユーザにより中断されたり、エラーが発生した場合、$1には0以外が代入されます。
    - バックアップがOn Backup Startupデータベースメソッド ($0 # 0)により停止された場合、$1には$0 引数で返された値が代入されます。これにより、独自のエラー管理システムを実装できます。
    - エラーのためバックアップが停止した場合、エラーコードが$1に返されます。

  いずれの場合もGET BACKUP INFORMATIONコマンドを使用してエラーに関する情報を入手できます。

注: データベースメソッドで$1 引数 (倍長整数) を宣言しなければなりません:

```
C_LONGINT($1)
```

On Backup Startupデータベースメソッド

On Backup Startupデータベースメソッドは、データベースのバックアップを開始しようとするたびに呼び出されます（手動でのバックアップ、定期的自動バックアップ、またはBACKUP コマンドによるバックアップ）。
これはすべての4D環境: 4D (すべてのモード), 4D Server、4D Volume Desktopが統合されたコンパイル済みアプリケーションに当てはまります。

On Backup Startupデータベースメソッドを使用すると、バックアップの開始を検証することができます。このメソッドからは、バックアップの許可または拒否を示す以下の値を引数$0にセットしてください:

- $0= 0:バックアップの開始を許可します。
- $0# 0:バックアップを拒否します。バックアップ処理はキャンセルされ、エラーが返されます。このエラーは、GET BACKUP INFORMATIONコマンドを使用して取得できます。

このデータベースメソッドを使用して、バックアップの実行条件を検証できます（ユーザ、前回のバックアップ日付など）。

Note: データベースメソッドで$0 引数 (倍長整数) を宣言しなければなりません:

```
C_LONGINT($0).
```

BACKUP

BACKUP
このコマンドは引数を必要としません

説明

BACKUPコマンドは、現在のバックアップ設定を用いて、データベースのバックアップを開始します。確認ダイアログボックスは表示されませんが、進捗バーが画面上に現れます。

バックアップは、データベース設定ダイアログ (4D v11 SQLでは環境設定) で設定します。これらの設定は、データベースのPreferences/backupサブフォルダー内にあるBackup.XMLファイルにも保存されます。

BACKUPコマンドは、実行開始時にOn Backup Startupデータベースメソッドを呼び出し、実行終了時にOn Backup Shutdownデータベースメソッドを呼び出します。
このメカニズムのため、このコマンドをこれらのデータベースメソッドから呼び出すべきではありません。

4D Server: クライアントマシンから呼び出された場合でも、**BACKUP**コマンドはストアドプロシージャーとみなされ、サーバー上で実行されます。

システム変数およびセット

バックアップが正しく実行されるとシステム変数OKに1が、そうでなければ0が設定されます。

エラー管理

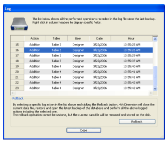
バックアップ中に問題が発生した場合、その問題に関する情報はバックアップログに書き込まれ、トップレベルのエラーだけがOn Backup Shutdownデータベースメソッドに送られます。プログラミングでバックアップエラーを管理するには、このデータベースメソッドを利用することが重要です。

CHECK LOG FILE

CHECK LOG FILE
このコマンドは引数を必要としません

説明

CHECK LOG FILE コマンドは、データベースのカレントログファイルの内容をブラウズできるダイアログを表示します。このダイアログにはMaintenance & Security Centerからもアクセスできます:



The list below shows all the performed operations recorded in the log file since the last backup.
Right click in column headers to display specific fields.

	Action	Table	User	Date	Hour
15	Addition	Table 3	Designer	3/22/2006	10:55:25 AM
16	Addition	Table 3	Designer	3/22/2006	10:55:26 AM
17	Addition	Table 3	Designer	3/22/2006	10:55:29 AM
18	Addition	Table 3	Designer	3/22/2006	10:55:37 AM
19	Addition	Table 4	Designer	3/22/2006	10:55:40 AM
20	Addition	Table 4	Designer	3/22/2006	10:55:41 AM
21	Addition	Table 4	Designer	3/22/2006	10:55:41 AM
22	Addition	Table 4	Designer	3/22/2006	10:55:42 AM
23	Addition	Table 4	Designer	3/22/2006	10:55:42 AM

By selecting a specific log action in the list above and clicking the Rollback button, 4th Dimension will close the current data file, restore and open the latest backup of the database and perform all the above logged actions including the selected one.
The rollback operation cannot be undone, but the current data file will be renamed and stored on the disk.

このダイアログ画面では、データベースのデータに対して行われた操作を取り消すことのできる、**ロールバック**ボタンがあります。このダイアログボックスの詳細については、4DのDesign Referenceを参照してください。

Note: ロールバックは相対的に強力な操作であるため、CHECK LOG FILEコマンドの使用はデータベースの管理者に制限することを推奨します。

このコマンドは、シングルユーザアプリケーションで実行している場合に使用できます。特に、このコマンドは4D Volume Desktopアプリケーション (デザインモードの無いアプリケーション) からロールバック機能へのアクセスを提供します。クライアント/サーバアプリケーションでこのコマンドを呼び出した場合、エラー1421が返され、コマンドは何も行いません。

エラー処理

- ログファイルを使用しないデータベースでこのコマンドを実行した場合、コマンドは何も行わず、エラー1403が返されます。
- このコマンドがクライアント/サーバデータベースで実行された場合、コマンドは何も行わず、エラー1421が返されます。
  ON ERR CALLコマンドでインストールされるエラー処理メソッドを使用して、これらのエラーをとらえることができます。

GET BACKUP INFORMATION

GET BACKUP INFORMATION (selector ; info1 ; info2)

引数	型		説明
selector	倍長整数	→	取得する情報のタイプ
info1	倍長整数, 日付	←	セレクタの値1
info2	時間, 文字	←	セレクタの値2

説明

GET BACKUP INFORMATION コマンドを使用して、データベースのデータに対して行われた前回のバックアップに関連する情報を取得できます。

selectorには取得する情報タイプを渡します。この引数の値として、“テーマ内の定数を使用できます:

定数	型	値
Last Backup Date	倍長整数	0
Last Backup Status	倍長整数	2
Next Backup Date	倍長整数	4

info1とinfo2の型および内容は、selectorの値によって決まります。

- selector = 0 (Last Backup Date): 前回のバックアップの、info1に日付、info2に時間が返されます。
- selector = 2 (Last Backup Status): 前回のバックアップの、info1にステータス番号、info2にそのテキストが返されます。
- selector = 4 (Next Backup Date): 次回のバックアップの、info1に日付、info2に時間が返されます。

GET RESTORE INFORMATION

GET RESTORE INFORMATION (selector ; info1 ; info2)

引数	型		説明
selector	倍長整数	→	取得する情報のタイプ
info1	倍長整数, 日付	←	セレクタの値1
info2	文字, 時間	←	セレクタの値2

説明

GET RESTORE INFORMATIONコマンドを使用し、前回のデータベース自動復元に関連する情報を取得できます。

selectorには取得する情報タイプを渡します。この引数の値として、“テーマ内の定数を使用できます:

定数	型	値
Last Restore Date	倍長整数	0
Last Restore Status	倍長整数	2

info1とinfo2の型および内容は、selectorの値によって決まります。

- selector = 0 (Last Restore Date): 前回の自動復元の、info1に日付、info2に時間が返されます。
- selector = 2 (Last Restore Status): 前回の自動復元の、info1にステータス番号、info2にそのテキストが返されます。

Note: このコマンドは手動でのデータベース復元は対象外となります。

INTEGRATE LOG FILE

INTEGRATE LOG FILE (pathName)

引数	型	説明
pathName	テキスト	→ 統合するログファイルのパス名

説明

INTEGRATE LOG FILEコマンドは、pathName引数に渡された名前やパス名のログファイルを、カレントデータベースに統合します。このコマンドは論理ミラーを使用したバックアップシステムを設定するために使用されます（4D Server Referenceの論理ミラーの設定を参照）。

アーカイブ化されていない、拡張子が.journalのファイルを、このコマンドで統合できます。ダイアログは表示されず、進捗バーが画面に表示されます。

pathName引数には、絶対パス名またはデータベースフォルダからの相対パス名を渡すことができます。空の文字列を渡すと、標準のファイルを開くダイアログボックスが表示され、ユーザはどのファイルを統合するか指定できます。このダイアログボックスをキャンセルした場合、ファイルは統合されず、システム変数OKには0が代入されます。

このコマンドを使用する場合、以下については開発者の判断で行ってください。

- ミラーマシンにミラーデータベースをインストールし、データファイルが**INTEGRATE LOG FILE**コマンドによるログファイルの統合以外で変更されることのないようにする。ミラー版のデータベースかどうかを調べるには、データベースフォルダーに任意のファイルを置き、例えばOn Startupデータベースメソッドなどでそのファイルが存在するかどうかをテストするなどします。ファイルが存在する場合、ミラーモードは有効になっていると判断できます。
- 操作中のデータベースとミラーデータベースの間に通信システムをセットアップし、ログファイルセグメントの送受信を管理する。このためにWebサービスまたは4D Internet Commandsを使用できます。
- 2つのデータベース間で起こりうる送信エラーを処理する。

システム変数およびセット

統合が正しく実行されるとシステム変数OKに1が、そうでなければ0が設定されます。

エラー管理

エラーが発生すると、コマンドは、ON ERR CALL コマンドでとらえることが可能なコードを生成します。ロックされたレコードがデータベースにあると、コマンドは何も行わず、エラー1420が生成されます。

INTEGRATE MIRROR LOG FILE

INTEGRATE MIRROR LOG FILE (pathName {; operationNum})

引数	型	説明
pathName	テキスト	→ 統合されるログファイルの名前もしくはパス名
operationNum	Real variable	→ 統合が開始されるオペレーションの番号 ← 最後に統合されたオペレーションの番号

説明

INTEGRATE MIRROR LOG FILE コマンドは、pathName で指定したログファイルの、任意の引数 operationNum 以降のオペレーションを、4D Server データベースへと統合します。このコマンドはどんなログファイルをもデータベースに統合することができます(たとえログファイルがカレントのデータファイルと対応していなくても 受け入れます)。このコマンドは特にミラーデータベースのコンテキストで使用することを目的としています。

注: 4D v14 以降、ログファイルを"ミラー"データベースの一部として使用することができます。"ログファイルを使用"のオプションは、論理ミラーとして使用されてい る4D Server のデータベース設定においてチェックが出来るようになりました。これにより、ミラーのミラーサーバーを実装することが出来るようになりました。

既存のINTEGRATE LOG FILE コマンドとは異なり、**INTEGRATE MIRROR LOG FILE** コマンドは実行を終了したあとにカレントログファイルを統合されたログファイルで置き換えることはしません。データベースのカレントログファイルは引き続き使用されます。これにより、統合の最中に変更されたものは全てカレントのログファイルへと記録されます。

pathName 引数には、データベースフォルダへの絶対パスまたは相対パスを渡します。この引数に空の文字列を渡した場合、標準のファイルを開くダイアログボックスが開 くので、そこから統合したいファイルを選択することができます。このダイアログボックスがキャンセルされると、どのファイルも統合されることなく、*OK* システム変数は0に設定されます。

デフォルトで、operationNum 引数を省略した場合、コマンドはログファイルの全てのオペレーションを統合します。
operationNum 引数に値を渡した場合、統合は指定された値のオペレーションから開始されます。この場合、統合が完了するとoperationNum 引数には最後に統合されたオペレーションの番号が渡されます。この返された値を次回**INTEGRATE MIRROR LOG FILE** コマンドを呼び出す際に直接渡す事で、連続したログファイル統合をすることができます。

以下のダイアグラム(pseudocode)は異なる場合における統合のプロセスの詳細を説明しています。統合されるログファイルにおいて、X は最初のオペレーションを、Y は最後のオペレーションを表す数値です:

```
Case of
    : (operationNum < X-1)
                        --> Error 1260 (Selected log file is too recent)
    : (operationNum >= X-1) and (operationNum <= Y)
                        --> Start integration
        Case of
            : (Type of operation = "Addition")
                If (Record exists)
                        --> Do nothing
                Else
                        --> Add record
                End if
            : (Type of operation = "Modification")
                If (Record exists)
                        --> Edit record
                Else
                        --> Do nothing
                End if
            : (Type of operation = "Deletion")
                If (Record exists)
                        --> Delete record
                Else
                        --> Do nothing
                End if
            End case
        Else// operationNum > Y
                        --> Error 1261 (Selected log file is too old for database)
End case
```

統合は、エラーが起こるとその時点で中止されてしまうことに注意して下さい。この場合、統合を続行したい場合にはMSCを使用する必要があります。

システム変数およびセット

統合が正しく実行されるとシステム変数OKに1が、そうでなければ0が設定されます。

Log File

Log File -> 戻り値

引数	型		説明
戻り値	文字	←	データベースログファイルのロング名

説明

Log Fileコマンドは、開いているデータベースのカレントログファイルのロング名（ファイル名を含むファイルの完全パス名）を返します。

データベースがログファイルを使用していない場合、コマンドは空の文字列を返し、システム変数OKには0が代入されます。

データベースがログファイルを使用している場合、システム変数OKには1が代入されます。コマンドに返されるパス名はカレントプラットフォームのシンタックスで表記されます。

警告: 4D Client マシンからこのコマンドを実行した場合、ロング名ではなくログファイル名のみが返されます。

システム変数およびセット

- データベースがログファイルなしで運用されている場合、システム変数OKは0に設定されます。そうでなければ1に設定されます。
- 実行中何らかの理由でログファイルが利用不能になった場合、エラー1274が生成され、4D Serverはデータの書き込みを一切許可しなくなります。ログファイルが利用可能になったのち、フルバックアップを行わなければなりません。

New log file

New log file -> 戻り値

引数	型		説明
戻り値	テキスト	↩	閉じられたログファイルのフルパス名

説明

注: このコマンドは4D Serverでのみ動作します。このコマンドはExecute on serverあるいはストアードプロシージャー内でのみ実行できます。

New log fileコマンドは、カレントログファイルを閉じて、そのファイルに別の名前を付けます。そして、以前と同じ名前の新しいログファイルを前のファイルと同じ場所に作成します。
このコマンドは論理ミラーを使用したバックアップシステムを設定するために使用されます（4D Server Referenceの論理ミラーの設定を参照）。

このコマンドは、閉じられたログファイルのフルパス名（アクセスパス+パス名）を返します。このファイルは、カレントログファイルと同じ場所に保存されます（これは環境設定のバックアップテーマ内、設定ページで指定します）。このコマンドは、保存するファイルに一切の処理（圧縮やセグメント化）を実行しません。ダイアログボックスは表示されません。

ファイルは、データベースやログファイルのカレントバックアップ番号を使用した別名に変更されます。次のようになります。
例: **DatabaseName[BackupNum-LogBackupNum].journal**

- MyDatabase.4DD というデータベースを4 回保存すると、最後のバックアップファイルの名前は、**MyDatabase[0004].4BK**となります。従って最初のログファイルの"セグメント"は、**MyDatabase[0004-0001].journal**です。
- MyDatabase.4DD というデータベースが3 回保存され、ログファイルが5 回保存された場合、6 度目に実行されるログファイルのバックアップは、**MyDatabase[0003-0006].journal**です。

エラー管理

エラーが発生すると、コマンドはON ERR CALL コマンドでとらえることが可能なエラーコードを生成します。

RESTORE

RESTORE {(archivePath {; destFolderPath})}

引数	型		説明
archivePath	テキスト	⇒	Pathname of archive to restore
destFolderPath	テキスト	⇒	Pathname of destination folder

説明

RESTORE コマンドは4Dアーカイブに含まれるファイルを復元するために使用できます。このコマンドはバックアップを管理するためのカスタマイズされたインターフェースで利用できます。

archivePath引数を渡さない場合、コマンドはファイルを開くダイアログを表示し、ユーザは復元するアーカイブを選択できます。

archivePath引数を使用すると、復元するアーカイブを指定できます。このパス名はシステムシンタックスで表現されなければなりません。絶対パス名またはストラクチャファイルからの相対パス名を指定できます。
この場合 (destFolderPath が省略されると)、アーカイブが選択された状態で標準の復元ダイアログボックスが表示され、ユーザーは保存先フォルダを選択できます。処理が完了すると警告ダイアログが表示され、復元された要素を含むフォルダが表示されます。

復元された要素の格納先フォルダのパス名としてdestFolderPath 引数を渡すこともできます。このパス名はシステムシンタックスで表現されなければなりません。絶対パス名またはストラクチャファイルからの相対パス名を指定できます。この引数を渡すと、設定済みの復元ダイアログボックスが表示され、ユーザは復元処理を起動するかキャンセルすることのみできます。処理が完了すると、追加の情報を表示することなしにウィンドウが閉じられます。

RESTORE コマンドは*OK*および*Document*変数を更新します。復元が正しく実行されると*OK*に1が設定され*Document*には復元要素保存先フォルダのパスが格納されます。復元ダイアログをユーザがキャンセルされたり、エラーが発生して復元が中断されたりすると、*OK*に0が設定され*Document*は空の文字列になります。ON ERR CALLコマンドを使用してインストールしたエラー処理メソッドを使用してエラーをとらえることができます。

Note: 4D Volume Desktopが統合されたコンパイル済みアプリケーションでは、RESTOREコマンドは標準のファイルオープンダイアログにデフォルトで"4BK"拡張子のファイルを表示します。

SELECT LOG FILE

SELECT LOG FILE (logFile | *)

引数	型		説明
logFile | *	演算子, 文字	→	ログファイルの名前、または * を指定するとカレントログを閉じる

説明

SELECT LOG FILE コマンドは、logFile引数で指定されたログファイルのオープン、作成、クローズを行います。

Note: SELECT LOG FILEを呼び出すことは、環境設定の**バックアップ/設定**ページで**ログファイルを使用**を選択あるいは選択解除するのと同じです。

logFileには、作成するログファイルの名前またはフルパス名を渡します。ファイル名のみが渡された場合、ファイルはデータベースのストラクチャファイルと同じ階層の"Logs"フォルダに作成されます。
logFileが空の文字列の場合、SELECT LOG FILEコマンドはファイルを保存ダイアログボックスを表示し、作成するログファイルの名前と場所を指定できます。ファイルが正しく作成されると、システム変数OKに1が設定されます。それ以外の場合、つまりユーザがキャンセルボタンをクリックしたり、ログファイルが作成できない場合は、システム変数OKに0が設定されます。

Note: 新しいログファイルはコマンドの実行後すぐには作成されません。次のバックアップ後に作成されます。パラメタはデータファイルに保持され、データベースが閉じられた後も有効です。BACKUP コマンドを呼び出して、ログファイルの作成をとりがすることもできます。

logFileに"*"を渡すと、SELECT LOG FILEコマンドはカレントログファイルを閉じます。ログファイルが閉じられると、システム変数OKに1が設定されます。

フルバックアップがまだ行われておらず、かつデータファイルに既にレコードが含まれているときに、SELECT LOG FILEコマンドを使ってログファイルを作成すると、4Dはエラー-4447を生成します。このエラーはON ERR CALLメソッドでとらえることができます。

システム変数およびセット

ログファイルが正しく作成されるか閉じられると、OKは1に設定されます。

エラー管理

データベースのバックアップが必要なため処理が行われなかった場合、エラー-4447が生成されます。このエラーはON ERR CALLメソッドでとらえることができます。

ピクチャ

- ピクチャ
- BLOB TO PICTURE
- COMBINE PICTURES
- CONVERT PICTURE
- CREATE THUMBNAIL
- Equal pictures
- Get picture file name
- GET PICTURE FROM LIBRARY
- GET PICTURE KEYWORDS
- GET PICTURE METADATA
- Is picture file
- PICTURE CODEC LIST
- PICTURE LIBRARY LIST
- PICTURE PROPERTIES
- Picture size
- PICTURE TO BLOB
- PICTURE TO GIF
- READ PICTURE FILE
- REMOVE PICTURE FROM LIBRARY
- SET PICTURE FILE NAME
- SET PICTURE METADATA
- SET PICTURE TO LIBRARY
- TRANSFORM PICTURE
- WRITE PICTURE FILE
- PICTURE TYPE LIST
- QT COMPRESS PICTURE
- QT COMPRESS PICTURE FILE
- QT LOAD COMPRESS PICTURE FROM FILE
- SAVE PICTURE TO FILE

ピクチャ

サポートされるネイティブフォーマット

4Dはピクチャフォーマットのネイティブ管理を統合します。これは4D内で、ピクチャが変換されることなく、元のフォーマットで格納、表示されることを意味します。(シェイドや透過など) フォーマットにより異なる特定の機能はコピーやペーストされる際にも保持され、変わることなく表示されます。このネイティブサポートは4Dに格納されるすべてのピクチャ (ライブラリピクチャ、デザインモードでフォームにペーストされたピクチャ、アプリケーションモードでフィールドや変数にペーストされたピクチャ) に対し有効です。4DはWindowsとMac OS両方でネイティブなAPIを使用して (フィールドや変数) ピクチャのエンコードやデコードを行います。これらの実装は (現在デジタルカメラで使用されている) RAWフォーマット含め、数多くのネイティブなフォーマットへのアクセスを提供します。

- **Windows**: 4DはWIC (Windows Imaging Component) を使用します。WICはネイティブに以下のフォーマットをサポートします: BMP, PNG, ICO (デコードのみ), JPEG, GIF, TIFF そして WDP (Microsoft Windows Digital Photo)。
  サードパーティーのWIC CODECをインストールしてJPEG-2000などの追加のフォーマットを使用できます。
- **Mac OS**: 4DはImageIOを使用します。利用可能なすべてのImageIO CODECはデコード (読み込み) およびエンコード (書き込み) をネイティブにサポートします:

デコード	**エンコード**
public.jpeg	public.jpeg
com.compuserve.gif	com.compuserve.gif
public.png	public.png
public.jpeg-2000	public.jpeg-2000
com.nikon.raw-image	public.tiff
com.pentax.raw-image	com.adobe.photoshop.image
com.sony.arw-raw-image	com.adobe.pdf
com.adobe.raw-image	com.microsoft.bmp
public.tiff com.canon.crw-raw-image	com.truevision.tga-image
com.canon.cr2-raw-image	com.sgi.sgi-image
com.canon.tif-raw-image	com.apple.pict
com.sony.raw.image	com.ilm.openexr-image
com.olympus.raw-image	
com.konicaminolta.raw-image	
com.panasonic.raw-image	
com.fuji.raw-image	
com.adobe.photoshop-image	
com.adobe.illustrator.ai-image	
com.adobe.pdf	
com.microsoft.ico	
com.microsoft.bmp	
com.truevision.tga-image	
com.sgi.sgi-image	
com.apple.quicktime-image	
com.apple.icns	
com.apple.pict	
com.apple.macpaint-image	
com.kodak.flashpix-image	
public.xbitmap-image	
com.ilm.openexr-image	
public.radiance	

Mac OSのようにWindowsでは、サポートされるフォーマットはOSおよびマシンにインストールされたカスタムCODECにより異なります。どのCODECが利用可能かを知るには、PICTURE CODEC LIST コマンドを使用します。

Note: WICおよびImageIOではピクチャのメタデータを利用できます。2つのコマンドSET PICTURE METADATAおよびGET PICTURE METADATAを使用してメタデータを利用した開発を行えます。

ピクチャ Codec ID

4Dが認識するピクチャフォーマットは PICTURE CODEC LIST コマンドからピクチャ Codec IDとして返されます。これは以下の形式で返されます:

- 拡張子 (例: ".gif")
- MIME タイプ (例: "image/jpeg")

それぞれのフォーマット用に返される形式は、CodecがOSレベルに記録された方法に基づきます。
多くの4Dピクチャ管理コマンドはCodec IDを引数として受けとることができます。したがって PICTURE CODEC LIST から返されるシステムIDを使用しなければなりません。

QuickTime の有効化(互換性)

デフォルトでは、QuickTimeに関連するピクチャコーデックは4D v14以降サポートされなくなっています。

互換性の理由から、QuickTimeをアプリケーション内で再度有効化することは可能です。SET DATABASE PARAMETER コマンドの QuickTime support オプションを利用して下さい。しかしながら、v14以降の QuickTime コーデックの使用は推奨されません。

ピクチャクリック時の座標

4Dではピクチャフィールドや変数をクリックした際のローカル座標を取得できます。これはスクロールやズームが行われている場合でも可能です。
クリック座標はMouseXとMouseYシステム変数に返されます。座標はピクセル単位で表現され、ピクチャの左上隅が (0,0) となります。この座標はOn Clicked または On Double Clickedフォームイベントで取得しなければなりません。このメカニズムが正しく動作するには、表示フォーマットが "**トランケート (中央あわせしない)**" に設定されていなければなりません (OBJECT SET FORMAT コマンド参照)。
ピクチャマップと同様のこの機能は、例えばスクロール可能なボタンバーや地図ソフトウェアのインタフェースを処理するために使用します。

ピクチャ演算子

4Dではピクチャの連結や重ね合わせなどのピクチャ操作を行うことができます。これはの節で説明されています。

BLOB TO PICTURE

BLOB TO PICTURE (pictureBlob ; picture {; codec})

引数	型		説明
pictureBlob	BLOB	→	ピクチャを格納したBLOB
picture	ピクチャー	←	BLOBから取り出したピクチャ
codec	文字	→	ピクチャcodec ID

説明

BLOB TO PICTURE コマンドは、元のフォーマットに関わらず、BLOBに格納されたピクチャを4Dのピクチャ変数やフィールドに挿入します。

このコマンドは READ PICTURE FILEコマンドと同様ですが、ファイルではなくBLOBに対して適用されます。このコマンドを使用すると、ネイティブのフォーマットでBLOBに保存されているピクチャを表示することができます。ピクチャのBLOBへのロードは、例えば DOCUMENT TO BLOB あるいは PICTURE TO BLOB コマンドを使用して行うことができます。

pictureBlob引数には、ピクチャを納めたBLOBタイプの変数やフィールドを渡します。このピクチャのフォーマットは4Dがネイティブにサポートされるものであればいずれの形式でも構いません。PICTURE CODEC LIST コマンドを使用すると、使用可能なフォーマットのリストを取得できます。任意のcodec 引数を渡すと、4Dはこの引数で指定された値を使用してBLOBをデコードします (この3番目の引数を使用した特別な機能については後述の説明を参照)。

picture 引数にはピクチャを表示する4Dピクチャフィールドまたは変数を渡します。

注: 内部的なピクチャフォーマットは4Dの変数やフィールドに格納されます。

コマンドが実行され、BLOBが正しくデコードされると、picture 引数には表示するピクチャが返されます。

オプションのcodec 引数を使用して、BLOBのデコード方法を指定できます。

codecに4Dが認識する (PICTURE CODEC LIST コマンドから返される) codecを渡すと、それはBLOBとピクチャフィールドや変数に返されるピクチャに適用されます。

codecに4Dが認識できないcodecを渡すと、新しいcodecが動的に引数に渡したIDで記録されます。そして4DはBLOBをカプセル化したピクチャを返し、OK変数に1を設定します。この場合BLOBを取り出すには、同じカスタムIDを使用してPICTURE TO BLOB コマンドを使用します。この特別なメカニズムは以下のような2つの特定のニーズで使用できます:

- ピクチャでないBLOBをカプセル化してピクチャに格納する
- codecを使用せずにピクチャをロードする

これらのメカニズムの実装は特に、ピクチャ配列を使用したBLOB配列の作成を可能にします。配列は全体がメモリにロードされるため、このメカニズムは注意 して使用されなければなりません。大きなサイズのBLOBで作業を行うと、アプリケーションの動作に影響を与えることがあります。

Note: VARIABLE TO BLOB コマンドで作成されたBLOBは自動で管理されます。BLOBは署名されるため、カプセル化するためにcodecを渡す必要はありません。この場合、反対の操作にはcodec IDとして".4DVarBlob"を PICTURE TO BLOB コマンドに渡します。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKは1に設定されます。変換に失敗 (QuickTimeがインストールされていない、Blobに読み取り可能なピクチャが格納されていない、codec引数を認識できたがBlobが有効でないなど) した場合、OKは0に設定され、4Dのピクチャ変数やフィールドは空になります。

COMBINE PICTURES

COMBINE PICTURES (resultingPict ; pict1 ; operator ; pict2 {; horOffset ; vertOffset})

引数	型		説明
resultingPict	ピクチャー	⇦	重ね合わせた結果のピクチャ
pict1	ピクチャー	⇨	重ね合わせる1つ目のピクチャ
operator	倍長整数	⇨	重ね合わせのタイプ
pict2	ピクチャー	⇨	重ね合わせる2つ目のピクチャ
horOffset	倍長整数	⇨	重ね合わせの横オフセット
vertOffset	倍長整数	⇨	重ね合わせの縦オフセット

説明

COMBINE PICTURES コマンドはpict1とpict2 ピクチャを、operatorモードで結合させ、3つめのresultingPictピクチャを得るために使用します。結果のピクチャは複合型で、ソースピクチャのすべての特性を保持します。

Note: このコマンドはピクチャ演算子 (+/等、の節参照) により提供される機能を拡張します。これらの演算子は4D v11でも引き続き利用できます。

operatorには適用する結合のタイプを指定します。"" テーマの定数にある3タイプの結合が使用できます:

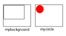
- Horizontal concatenation (1): pict2がpict1に追加され、pict2の左上隅がpict1の右上隅の位置に置かれます。
- Vertical concatenation (2): pict2がpict1に追加され、pict2の左上隅がpict1の左下隅の位置に置かれます。
- Superimposition (3): pict2はpict1の上に置かれ、pict2の左上隅がpict1の左上隅の位置に置かれます。

オプションのhorOffsetとvertOffset引数が使用されると、重ね合わせの前に pict2 に対する解釈が適用されます。horOffset と vertOffsetに渡される値はピクセル単位でなければなりません。正数を渡すと右あるいは下方向へのオフセット、負数を渡すと左あるいは上方向へのオフセットとなります。

Note: COMBINE PICTURES コマンドで実行される重ね合わせは、& と | 演算子で提供される (ORやXORの) 重ね合わせとは異なります。COMBINE PICTURES コマンドは結果ピクチャ中にソースピクチャの特性を保持しますが、& と | 演算子はそれぞれのピクセルを処理してビットマップピクチャを生成します。これらの演算子はもともと白黒画像を処理するためのものであり、現在は廃止予定です。

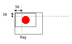
例題

以下のピクチャがある時:



```
COMBINE PICTURES(flag;mybackground;Superimposition;mycircle;50;30)
```

結果は:



CONVERT PICTURE

CONVERT PICTURE (picture ; codec {; compression})

引数	型		説明
picture	ピクチャー	⇒	変換するピクチャ
		⇐	変換されたピクチャ
codec	文字	⇒	ピクチャCodec ID
compression	実数	⇒	圧縮の品質

説明

CONVERT PICTURE コマンドはpictureを新しいタイプに変換します。

codec 引数で生成するピクチャのタイプを指定します。Codecには拡張子 (例 ".gif"), またはMimeタイプ(例 "image/jpeg")が使用できます。利用可能なCodecのリストは PICTURE CODEC LIST コマンドを使用して取得できます。

picture フィールドや変数が複合型 (例えばコピー/ペーストアクションの結果のピクチャ) の場合、codecタイプに対応する情報のみが結果ピクチャに保持されます。

Note: リクエストされた codec のタイプがpictureの元のタイプと同じ場合、変換は実行されず、そのままのピクチャが返されます (compression引数が使用された場合を除く、後述参照)。

オプションのcompression引数が渡されると、互換性のあるCODEC利用時に、結果のピクチャに適用する圧縮品質を指定できます。compressionに は圧縮品質を指定する0から1の間の値を渡します。0が圧縮優先で1が品質優先です。この引数はCODECが圧縮をサポートし (例えばJPEGやHDPhoto)、さらにWICやImageIO APIによりサポートされている場合にのみ考慮されます。ピクチャ管 理APIに関する詳細情報は、ピクチャ を参照してください。compression引数を省略するとデフォルトで品質優先 (compression =1) が適用されます。

例題 1

vpPhoto ピクチャをjpegフォーマットに変換:

```
CONVERT PICTURE(vpPhoto;".jpg")
```

例題 2

60%の品質でピクチャを変換:

```
CONVERT PICTURE(vPicture;".JPG";0.6)
```

CREATE THUMBNAIL

CREATE THUMBNAIL (source ; dest {; width {; height {; mode {; depth}}}})

引数	型		説明
source	ピクチャー	⇒	サムネイルに変換する4Dピクチャフィールド または 変数
dest	ピクチャー	⇐	結果のサムネイル
width	整数	⇒	サムネイル幅 (ピクセル), デフォルト値 = 48
height	整数	⇒	サムネイル高 (ピクセル), デフォルト値 = 48
mode	整数	⇒	サムネイル作成モード デフォルト値 = Scaled to fit prop centered (6)
depth	整数	⇒	廃止。使用しないでください

説明

CREATE THUMBNAILコマンドは、指定した元のピクチャのサムネールを返します。通常、サムネールはマルチメディアソフトウェアやWebサイトにおいてピクチャプレビューのために使用されます。

source引数にはサムネールに縮小するピクチャが入った4Dの変数またはフィールドを渡します。dest引数には結果のサムネイルを受け取る4Dのピクチャフィールドまたはピクチャ変数を渡します。

オプションの引数widthおよびheightを使用し、必要とするサムネールのサイズ（ピクセル単位）を指定できます。この2つの引数を省略すると、サムネールのデフォルトサイズは48 x 48ピクセルになります。

オプションの引数modeには、サムネールの作成モード、例えばリサイズモードを指定できます。3種類のモードが使用できます。以下の定義済み定数が、定数テーマPicture Display Formatsで提供されています:

定数	型	値
Scaled to Fit	倍長整数	2
Scaled to fit prop centered	倍長整数	6
Scaled to fit proportional	倍長整数	5

注: **CREATE THUMBNAIL**ではこれら3つの定数のみが使用できます。このテーマの他の定数はこのコマンドに適用できません。

この引数を指定しない場合、"Scaled to fit prop centered" モード (6) がデフォルトで適用されます。モードごとの結果を以下に示します:

ソースピクチャ

結果のサムネイル (48x48)

- Scaled to fit = 2

- Scaled to fit proportional = 5

- Scaled to fit prop centered = 6 (デフォルトモード)

注: Scaled to fit proportional および Scaled to fit prop centeredを使用すると、空いたスペースが白く表示されます。しかし、これらのモードが4Dフォームのピクチャフィールドまたはピクチャ変数に適用されると、この空きスペースは透明になります。

depth 引数は無視され、省略されなければなりません。コマンドは常に現在のスクリーン深度 (色数) を使用します。

Equal pictures

Equal pictures (picture1 ; picture2 ; mask) -> 戻り値

引数	型		説明
picture1	ピクチャーフィールド, ピクチャー変数	→	元のソースピクチャー
picture2	ピクチャーフィールド, ピクチャー変数	→	比較するピクチャー
mask	ピクチャーフィールド, ピクチャー変数	←	結果のマスク
戻り値	ブール	↩	2つのピクチャーが同じ場合True、そうでなければFalse

説明

Equal picturesコマンドは2つのピクチャーの寸法と内容を厳密に比較します。

picture1 には元のピクチャーを、picture2 には比較したいピクチャーを渡します。

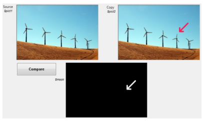
- ピクチャーの寸法が異なる場合、コマンドは**False**を返し、mask 引数には空のピクチャーが返されます。
- ピクチャーの寸法が同じで内容が異なる場合、コマンドは**False**を返し、mask 引数には2つのピクチャーを比較したピクチャーマスクの結果が返されます。この比較はピクセルごとに行われ、黒の背景上に、一致しないピクセルが白で表されます。
- 両ピクチャーがまったく同じ場合、コマンドは**True**を返し、mask 引数には黒のピクチャーが返されます。

システム変数およびセット

コマンドが正しく実行（2つのピクチャが比較）されると、システム変数OKは1に設定されます。そうでなければOKに0が設定されます。具体例として、片方のピクチャが初期化されていない（空のピクチャ）であると、システム変数OKは0に設定されます。

例題

この例題では2つのピクチャー (pict1 と pict2) を比較し、結果のマスクを取得します:



以下は比較を実行するコードです:

```
 $equal :=Equal pictures($pict1;$pict2;$mask)
```

Get picture file name

Get picture file name (picture) -> 戻り値

引数	型		説明
picture	ピクチャーフィールド, ピクチャー変数	→	デフォルト名を取得するピクチャー
戻り値	テキスト	←	ピクチャーファイルのデフォルト名

説明

Get picture file nameコマンドは引数に渡されたピクチャーのカレントデフォルト名を返します。

デフォルト名はピクチャーをディスクファイルに書き出す際に使用されます。この名前はピクチャーフィールドや変数にピクチャーファイルを読み込んだ時の元の名前あるいはSET PICTURE FILE NAMEコマンドにより設定されているかもしれません。詳細はデザインリファレンスマニュアルを参照してください。

ピクチャーにデフォルト名が設定されていない場合、コマンドは空の文字列を返します。

GET PICTURE FROM LIBRARY

GET PICTURE FROM LIBRARY (picRef | picName ; picture)

引数	型		説明
picRef | picName	倍長整数, 文字	→	ピクチャライブラリ画像の参照番号 または ピクチャライブラリ画像の名前
picture	ピクチャー変数	←	ピクチャライブラリのピクチャ

説明

GET PICTURE FROM LIBRARY コマンドは、picRefに渡された参照番号またはpicNameに渡された名前を持つピクチャライブラリの画像をpicture引数に返します。

参照番号または名前に対応するピクチャがない場合、GET PICTURE FROM LIBRARYコマンドはpictureを変更しません。

例題 1

以下の例は、参照番号がローカル変数$vlPicRef変数に格納されたピクチャをvgMyPicture変数に返します:

```
GET PICTURE FROM LIBRARY($vlPicRef;vgMyPicture)
```

例題 2

次の例では$DDcom_Prot_MyPictureに、ピクチャライブラリ中に保存されている"DDcom_Prot_Button1"という名前の画像を返します:

```
GET PICTURE FROM LIBRARY("DDcom_Prot_Button1";$DDcom_Prot_MyPicture)
```

例題 3

PICTURE LIBRARY LISTコマンドの例題参照

システム変数およびセット

ピクチャライブラリが存在すればOK変数に1が設定され、そうでなければ0が設定されます。

エラー管理

ピクチャに返すための十分なメモリがない場合、エラーコード-108が生成されます。エラー処理メソッドを使って、このエラーを受け取ることができます。

GET PICTURE KEYWORDS

GET PICTURE KEYWORDS (picture ; arrKeywords {; *})

引数	型		説明
picture	ピクチャーフィールド, ピクチャー変数	→	割り当てられたキーワードを取得するピクチャー
arrKeywords	テキスト配列	←	キーワードを受け取る配列
*	演算子	→	指定時: 重複するキーワードを取り除く

説明

GET PICTURE KEYWORDSコマンドは引数に渡したピクチャーに割り当てられたキーワードのリストをarrKeywords配列に返します。

IPTC/Keywords メターデータを使用して設定されたキーワードだけが返されます。他のタイプのメタデータはこのコマンドから無視されます。このコマンドは、このタイプのメタデータをサポートするピクチャー (JPEG, TIFF, 等) に対してのみ動作します。

注: 4DはIPTC/Keywordsタイプのメタデータをインデックスすることが可能です (デザインリファレンスマニュアルを参照)。

* 引数を渡すと、コマンドはキーワードの重複しない値のみを返します。つまりリスト中のすべての値がユニークになります。

ピクチャーにIPTC/Keywordsメターデータが含まれない場合、コマンドは空の配列を返し、エラーは生成されません。

注: このコマンドから返される結果はデータベース設定の"非文字・非数字のみをキーワード区切り文字とする"の現在値により異なります (データベース/データストレージページ参照)。

GET PICTURE METADATA

GET PICTURE METADATA (picture ; metaName ; metaContents {; metaName2 ; metaContents2 ; ... ; metaNameN ; metaContentsN})

引数	型		説明
picture	ピクチャー	⇒	メタデータを読むピクチャ
metaName	テキスト	⇒	取得するブロックの名前またはパス
metaContents	変数	⇐	メタデータの内容

説明

GET PICTURE METADATAコマンドを使用して picture で指定したピクチャ (4Dのピクチャフィールドや変数) 内のメタデータ (またはメタタグ) の内容を読みだすことができます。メタデータに関する詳細はSET PICTURE METADATAコマンドの説明を参照してください。

metaName引数には取り出すメタデータのタイプを指定する文字列を渡します。以下を渡すことができます:

- タグパスが含まれる Picture Metadata Names テーマの定数
- メタデータの完全ブロック名 ("TIFF", "EXIF", "GPS" あるいは"IPTC")。
- 空の文字列 ("")。

metaContents 引数にはメタデータを受け取る変数を渡します。

- metaNameにタグパスを渡した場合、metaContents 引数は直接取得した値を含みます。値は変数の型に合わせて変換されます。テキスト型の変数ではXML (XMP標準) でフォーマットされます。(特にIPTC Keywordsタグのように) メタデータに一つ以上の値が含まれる場合、配列を渡すことができます。
- metaNameにブロック名か空の文字列を渡すとき、metaContents 引数は有効なXML DOM要素参照でなければなりません。この場合、指定されたブロック (あるいはmetaNameに空の文字列を渡した場合はすべてのブロック) の内容は参照された要素に再コピーされます。

例題 1

DOMツリーストラクチャを使用する

```
$xml:=DOM Create XML Ref("Root") // XML DOMツリーの作成

 // TIFFメタデータの読み出し
$_Xml_TIFF:=DOM Create XML element($xml;"/Root/TIFF")
GET PICTURE METADATA(vPicture;"TIFF";$_Xml_TIFF)

 // GPSメタデータの読み出し
$_Xml_GPS:=DOM Create XML element($xml;"/Root/GPS")
GET PICTURE METADATA(vPicture;"GPS";$_Xml_GPS)
```

例題 2

変数の使用

```
C_DATE($dateAsDate)
GET PICTURE METADATA(vPicture;TIFF Date Time;$dateAsDate)
 // $dateAsDateが日付型のため、日付のみが返される

C_TEXT($dateAsText)
GET PICTURE METADATA(vPicture;TIFF Date Time;$dateAsText)
 // 日付のみがXMLフォーマットで返される

C_INTEGER($urgency)
GET PICTURE METADATA(vPicture;IPTC Urgency;$urgency)
```

例題 3

タグの複数の値が配列に返される

```
ARRAY TEXT($tTkeywords;0)
GET PICTURE METADATA(vPicture;IPTC Keywords;$tTkeywords)
```

コマンドの実行後、$arrTkeywordsは例えば以下のようになります:

```
$arrTkeywords{1}="France"
$arrTkeywords{2}="Europe"
```

例題 4

テキスト変数に複数の値を持つタグを受信する

```
C_TEXT($vTwords;0)
GET PICTURE METADATA(vPicture;IPTC Keywords;$vTwords)
```

コマンド実行後、$vTwordsは例えば"France;Europe"のようになります。

システム変数およびセット

メタデータの取得が正しく行われると、*OK*システム変数に1が設定され、エラーが発生したり1つ以上のタグが見つからない場合は0が設定されます。どのような場合でも、読みだされた値は返されます。

Is picture file

Is picture file (filePath {; *}) -> 戻り値

引数	型		説明
filePath	テキスト	⇒	ファイルパス名
*	演算子	⇒	データの検証
戻り値	ブール	↩	True = filePathはピクチャファイルである、そうでなければFalse

説明

Is picture file コマンドはfilePath引数で指定されたファイルをテストし、有効なピクチャファイルであればTrueを返します。ファイルがピクチャファイルでないか、ファイルが見つからない場合はFalseを返します。

filePath引数にはテストするピクチャファイルのパス名を渡します。パスはシステムシンタックスで指定しなければなりません。絶対パスまたははデータベースストラクチャファイルに対する相対パスを渡すことができます。空の文字列 ("") を渡すと、コマンドはFalseを返します。

* 引数を渡さない場合、コマンドはファイルの拡張子と利用可能なコーデックのリストを照合することでテストを行います。拡張子なしやより詳細なテストを行いたい場合は、* 引数を渡します。この場合、コマンドは追加のテストを行います。つまりファイルヘッダをロード・検査し、コーデックを照合してピクチャの妥当性を検査します。このシンタックスはコマンド実行を遅くします。

Note: コマンドはWindowsでPDFファイルに対し、Mac OSでEMFファイルに対し、Trueを返します。

PICTURE CODEC LIST

PICTURE CODEC LIST (codecArray {; namesArray}{; *})

引数	型		説明
codecArray	文字配列	⇐	利用可能なピクチャCodecのID
namesArray	文字配列	⇐	ピクチャCodecの名前
*	演算子	⇒	読み込み (デコード) CODECのリストを返す

説明

PICTURE CODEC LIST コマンドは、コマンドが実行されたマシンで利用可能なピクチャCodec IDのリストをcodecArray 配列に返します。このリストは4DがネイティブでI管理するCodec IDを含みます。

Codec IDは以下の異なるフォーマットでcodecArray 配列に返されます:

- 拡張子 (例: ".gif")
- MIME タイプ (例: "image/jpeg")

互換性に関する注意: QuickTimeがデータベース内で有効になっている場合(ピクチャ を参照)、4文字の QuickTimeコードが返されることもあります (例: "PNTG")。

コマンドから返されるフォーマットは、CodecがOSレベルに記録された方法に基づきます。オプションのnamesArray 配列を使用してそれぞれのCodecの名称を取得できます。この名称はIDよりも明確です。この配列は例えば利用可能なCodecのメニューリストを作成するために使用できます。

* 引数を渡さないとデフォルトでコマンドはピクチャをエンコード (書き込み) するために使用できるCODECのみを返します。これらのIDはピクチャ書き出しコマンド WRITE PICTURE FILE や PICTURE TO BLOB のformat 引数で使用できます。
* 引数を渡すと、コマンドはピクチャのデコード (読み込み) に使用するCODECも返します。2つのリストは排他的ではありません。特定の読み込みおよび書き込みCODECは同じです。ピクチャのエンコードを意図 するCODECは通常デコードに使用されます。他方デコード用のCODECは必ずしもエンコードに使用できるとは限りません。例えば".jpg" CODECは両方のリストにありますが、".xbmp"CODECは読み込み (デコード) CODECにしかありません。

PICTURE LIBRARY LIST

PICTURE LIBRARY LIST (picRefs ; picNames)

引数	型		説明
picRefs	倍長整数配列	←	ピクチャライブラリ画像の参照番号
picNames	文字配列	←	ピクチャライブラリ画像の名前

説明

PICTURE LIBRARY LIST コマンドは、データベースのピクチャライブラリの中に現在格納されているピクチャの参照番号と名前を返します。

このコマンドを呼び出すと、picRefs配列の中に参照番号、picNames配列の中にピクチャ名が返されます。この2つの配列は同期します。つまりpicRefs配列のn番目の要素は、ピクチャライブラリ内でpicNames配列のn番目の要素内に返されるピクチャ名が持つ参照番号になります。

必要であればコマンドは自動でpicRefsとpicNames配列を作成しサイズ調整します。

ピクチャライブラリのピクチャの名前は最大255文字です。

ピクチャライブラリの中にピクチャがない場合、両方の配列は空で返されます。

ピクチャライブラリの中に現在格納されているピクチャの数を取得するには、Size of arrayコマンドを使って、2つの配列のどちらかのサイズを取得します。

例題 1

以下のコードは、配列alPicRefとasPicNameの中にピクチャライブラリのカタログを返します:

```
PICTURE LIBRARY LIST(alPicRef;asPicName)
```

例題 2

以下の例は、ピクチャライブラリが空であるかどうかを検査します:

```
PICTURE LIBRARY LIST(alPicRef;asPicName)
If(Size of array(alPicRef)=0)
   ALERT("The Picture Library is empty.")
Else
   ALERT("The Picture Library contains "+String(Size of array(alPicRef))+" pictures.")
End if
```

例題 3

以下の例は、ピクチャライブリをディスク上のドキュメントに書き出します::

```
PICTURE LIBRARY LIST($alPicRef;$asPicName)
$vlNbPictures:=Size of array($alPicRef)
If($vlNbPictures>0)
   SET CHANNEL(12;"")
   If(OK=1)
      $vsTag:="4DV6PICTURELIBRARYEXPORT"
      SEND VARIABLE($vsTag)
      SEND VARIABLE($vlNbPictures)
      gError:=0
      For($vlPicture;1;$vlNbPictures)
         $vlPicRef:=$alPicRef{$vlPicture}
         $vsPicName:=$asPicName{$vlPicture}
         GET PICTURE FROM LIBRARY($alPicRef{$vlPicture};$vgPicture)
         If(OK=1)
            SEND VARIABLE($vlPicRef)
            SEND VARIABLE($vsPicName)
            SEND VARIABLE($vgPicture)
         Else
            $vlPicture:=$vlPicture+1
            gError:=-108
         End if
      End for
      SET CHANNEL(11)
      If(gError#0)
         ALERT("The Picture Library could not be exported, retry with more memory.")
         DELETE DOCUMENT(Document)
      End if
   End if
Else
   ALERT("The Picture Library is empty.")
End if
```

PICTURE PROPERTIES

PICTURE PROPERTIES (picture ; width ; height {; hOffset {; vOffset {; mode}}})

引数	型		説明
picture	ピクチャー	→	情報を取得するピクチャ
width	倍長整数	←	ピクチャの幅 (ピクセル)
height	倍長整数	←	ピクチャの高さ (ピクセル)
hOffset	倍長整数	←	バックグランド表示の時の水平方向のオフセット
vOffset	倍長整数	←	バックグランド表示の時の垂直方向のオフセット
mode	倍長整数	←	バックグランド表示の時の転送モード

説明

PICTURE PROPERTIES コマンドは、pictureに渡したピクチャに関する情報を返します。

引数widthとheightには、ピクチャの幅と高さを返します。

オプション引数hOffset、vOffset、modeには、フォーム上でバックグラウンド 表示フォーマットで表示された際のピクチャの水平、垂直位置と転送モードを返します。

Picture size

Picture size (picture) -> 戻り値

引数	型		説明
picture	ピクチャー	→	サイズを知りたいピクチャ
戻り値	倍長整数	←	ピクチャのサイズ (バイト)

説明

Picture sizeコマンドは指定したピクチャのサイズをバイト単位で返します。

PICTURE TO BLOB

PICTURE TO BLOB (picture ; pictureBlob ; codec)

引数	型		説明
picture	ピクチャー	→	ピクチャフィールドまたは変数
pictureBlob	BLOB	←	変換されたピクチャを受け取るBLOB
codec	文字	→	ピクチャCodec ID

説明

PICTURE TO BLOB コマンドは、4D変数やフィールドに格納されたピクチャを他のフォーマットに変換し、変換後のピクチャをBLOB内に納めます。

ピクチャタイプの4Dフィールドや変数をpicture引数に渡します。pictureBlob引数には、変換後のピクチャを納めるBLOB変数やフィールドを渡します。

codec 引数には変換フォーマットを指定する文字列を渡します。
Codecは拡張子 (例 ".gif"), または Mimeタイプ (例 "image/jpeg") です。利用可能なCodecは PICTURE CODEC LIST コマンドで取得できます。

このコマンドを実行すると、picturBlobには指定したフォーマットのピクチャが納められます。

変換が正常に終了すると、システム変数OKには1が代入されます。変換が失敗した(変換できない)場合、システム変数OKに0が代入され、生成されたBLOBは空です(0バイト)。

PICTURE TO GIF

PICTURE TO GIF (pict ; blobGIF)

引数	型		説明
pict	ピクチャー	→	ピクチャフィールドまたは変数
blobGIF	BLOB	←	GIFピクチャを受け取るBLOB

説明

PICTURE TO GIFコマンドは、変数またはフィールドに保存されているPICTピクチャーから、GIFフォーマットの画像を作成します。

ピクチャータイプの変数またはフィールドをpictに、BLOBタイプの変数またはフィールドをblobGIFに渡します。コマンドの実行後、blobGIFにはGIFフォーマットの画像が格納されます。

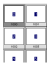
注: GIFピクチャーは256色以上をサポートしません。元のPICTピクチャーがそれ以上の色数の場合、失われる色があります。コマンドはシステムパレットにもとづいて減色します。生成されるGIF画像は87a (opaque) およびノーマル (インターレースなし)です。

blobGIFの画像をBLOB TO DOCUMENTコマンドを使ってファイルに保存でき、またそれをWeb上に公開することもできます。

変換が成功したらシステム変数OKが1になります。そうでない場合は0になります。

例題

接続カウンターを表示するGIF画像を作成する場合を想定してみましょう。データベースのピクチャライブラリ内に、すべての番号をピクチャーとして登録しておきます:



On Web Connectionデータベースメソッドに、以下のコードを記述します:

```
C_BLOB($blob)
Case of
      ...
   :($1="/4DCGI/counter") // カウンターをGIF画像で生成
 // 4DにこのURLが送信されると
      $blob:=gifcounter(<>nbHits) // GIF画像で生成
 // <>nbHits変数には接続数がおさめられている
      WEB SEND BLOB($blob;"image/gif")
 // 生成したピクチャーをブラウザーに送信
      ...
End case
```

以下はgifcounter メソッドです:

```
C_LONGINT($1)
C_PICTURE($img)
C_BLOB($0)
If($1=0)
   $ndigits:=1
Else
   $ndigits:=1+Length(String($1))
End if
If($ndigits<5)
   $ndigits:=5
End if

$div:=10^($ndigits-1)
For($i;1;$ndigits)
   $ref:=Int($1/$div)%10
   GET PICTURE FROM LIBRARY($ref+1000;picture)
   $img:=$img+picture
   $div:=$div/10
End for

PICTURE TO GIF($img;$0)
```

4DはWebブラウザーに以下のようなGIFピクチャーを送信します:

システム変数およびセット

変換に成功するとシステム変数OKに1が、そうでなければ0が設定されます。

READ PICTURE FILE

READ PICTURE FILE (fileName ; picture {; *})

引数	型		説明
fileName	文字	⇒	読み込むファイルのフルパス名, または空の文字列
picture	ピクチャー	⇐	ピクチャを受け取るフィールドまたは変数
*	演算子	⇒	指定時 = すべてのファイルタイプを受け入れる

説明

READ PICTURE FILE コマンドを使用してディスクファイルfileName に保存されたピクチャを開き、これをpicture 引数に指定した4Dフィールドまたは変数へロードすることができます。

fileName には読み込むファイルのフルパス名またはファイル名のみを渡すことができます。ファイル名のみを渡した場合、そのファイルはデータベースストラクチャと同階層になければなりません。Windowsではファイル拡張子が必要です。
空の文字列 ("") がfileName に渡されると、標準のファイルを開くダイアログボックスが表示され、ユーザは読み込むファイルやフォーマットを指定できます。

PICTURE CODEC LIST コマンドを使用して、利用可能なフォーマットを取得できます。

pictureには、読み込んだピクチャを受け取るピクチャ変数またはフィールドを渡します。

Note: 内部的なピクチャフォーマットは4D変数やフィールドに格納されます。

オプションの * 引数を渡すと、コマンドはすべてのタイプのファイルを受け入れます。この方法では、適切なcodecなしでもピクチャを扱うことができます(BLOB TO PICTURE コマンドの説明を参照してください)。

システム変数およびセット

コマンドが正しく実行されると、システム変数Documentには開かれたファイルのフルパスが格納され、システム変数OKは1に設定されます。そうでなければOKに0が設定されます。

REMOVE PICTURE FROM LIBRARY

REMOVE PICTURE FROM LIBRARY (picRef | picName)

引数	型		説明
picRef | picName	倍長整数, 文字	→	ピクチャライブラリ画像の参照番号 または ピクチャライブラリ画像の名前

説明

REMOVE PICTURE FROM LIBRARY コマンドは、picRef引数に渡した参照番号またはpicName引数の名前を持つピクチャをピクチャライブラリから消去します。

参照番号または名前を持つピクチャがない場合は、このコマンドは何も行いません。

4D Server: REMOVE PICTURE FROM LIBRARYコマンドはサーバマシン上で実行される (ストアードプロシージャやトリガ) メソッドの中から使用することはできません。REMOVE PICTURE FROM LIBRARYコマンドをサーバマシン上で呼び出しても、無視され、何も行われません。

警告: デザインオブジェクト (階層リスト項目、メニュー項目等) は、ピクチャライブラリのピクチャを参照することができます。プログラムによってピクチャライブラリのピクチャを修正する際は、注意する必要があります。

例題 1

以下の例は、ピクチャライブラリから参照番号4444のピクチャを削除します。

```
REMOVE PICTURE FROM LIBRARY(4444)
```

例題 2

以下の例は、ドル記号 ($) で始まる名前を持つピクチャをピクチャライブラリから削除します:

```
PICTURE LIBRARY LIST($alPicRef;$asPicName)
For($vlPicture;1;Size of array($alPicRef))
   If($asPicName{$vlPicture}="$@")
      REMOVE PICTURE FROM LIBRARY($alPicRef{$vlPicture})
   End if
End for
```

SET PICTURE FILE NAME

SET PICTURE FILE NAME (picture ; fileName)

引数	型		説明
picture	ピクチャーフィールド, ピクチャー変数	⇒	デフォルト名を設定するピクチャー
fileName	テキスト	⇒	デフォルトピクチャー名

説明

SET PICTURE FILE NAMEコマンドは引数に渡したピクチャーのデフォルト名を設定あるいは変更します。

デフォルト名はピクチャーをディスクファイルに書き出す際のデフォルト名として使用されます。フィールドの内容が変数や他のフィールドにコピーされると、デフォルト名もコピーされます。詳細はデザインリファレンスマニュアルを参照してください。

SET PICTURE METADATA

SET PICTURE METADATA (picture ; metaName ; metaContents {; metaName2 ; metaContents2 ; ... ; metaNameN ; metaContentsN})

引数	型		説明
picture	ピクチャー	⇨	メタデータを設定するピクチャ
metaName	テキスト	⇨	設定するブロックの名前またはパス
metaContents	変数	⇨	メタデータの内容

説明

SET PICTURE METADATA コマンドを使用して、picture (4Dピクチャフィールドまたは変数) 内のメタデータ (またはメタタグ) の内容を書き込んだり更新したりできます。

メタデータはピクチャに挿入された追加の情報です。4Dでは4タイプの標準メタデータEXIF, GPS, IPTC そして TIFFを処理できます。

Note: これらのメタデータタイプについては以下のドキュメントを参照できます: http://www.iptc.org/std/IIM/4.1/specification/IIMV4.1.pdf (IPTC) および http://exif.org/Exif2-2.PDF (TIFF, EXIF, GPS).

metaName引数には設定や更新を行うメタデータタイプを指定する文字列を渡します。以下を渡すことができます:

- 新しいPicture Metadata Namesテーマの定数。このテーマには4Dがサポートするすべてのタグがグループ化されています。それぞれの定数はタグパスを含んでいます (例 "TIFF/DateTime")。
- メタデータの完全なブロック名 ("TIFF", "EXIF", "GPS" あるいは "IPTC")。
- 空の文字列 ("")。

metaContents 引数にはメタデータの新しい値を渡します:

- metaNameにタグパス定数を渡した場合、設定する値を直接、または新しいPicture Metadata Valuesテーマの適切な定数を、metaContents引数に渡せます。値は指定されたメタデータに応じて、テキスト、倍長整数、実数、日付、時間タイプを使用できます。メタデータが一つ以上の値を含む場合、配列を使用できます。文字列を渡すときはXML (XMP標準) でフォーマットされていなければなりません。空の文字列 ("") を渡すと既存のメタデータが消去されます。
- metaNameにブロック名か空の文字列を渡すときは、metaContents引数には設定するメタデータを含むXML DOM参照を渡します。空の文字列の場合、すべてのメタデータが更新されます。

Windowsではコマンド実行時にエラーが発生すると、*OK* 変数が0に設定されます。Mac OSでは技術的な理由で、メタデータ書き込みエラーを検知できません。そのためこのコマンドはMac OSでは*OK* 変数を更新しません。

警告: 特定のメタデータはリードオンリーであり、SET PICTURE METADATAでは書き込むことはできません。例えば、TIFF XResolution/TIFF YResolution、EXIF Color Space、EXIF Pixel X Dimension/EXIF Pixel Y Dimensionなどは修正できません。

注: 特定のピクチャーフォーマット (特にJPEGとTIFF) だけがメタデータをサポートします。逆にGIFやBMPなどのフォーマットはメタデータを受け入れません。メタデータ付きのピクチャーを、それをサポートしないフォーマットに変換すると、情報は失われます。

例題 1

配列を使用して"Keywords"メタデータの値を複数設定します:

```
ARRAY TEXT($arrTkeywords;2)
$arrTkeywords{1}:="France"
$arrTkeywords{2}:="Europe"
SET PICTURE METADATA(vPicture;IPTC Keywords;$arrTkeywords)
```

例題 2

DOM参照を使用してGPSブロックを設定します:

```
C_TEXT($domMetas)
$domMetas:=DOM Parse XML source("metas.xml")
C_TEXT($gpsRef)
$gpsRef:=DOM Find XML element($domMetas;"Metadatas/GPS")
If(OK=1)
   SET PICTURE METADATA(vImage;"GPS";$refGPS)
 //ここで $gpsRef はGPS要素を指しています
   ...
End if
DOM CLOSE XML($domMetas)
```

注

すべてのメタデータがDOM要素参照で処理されるとき、タグは名前がブロック名 (TIFF, IPTC等)である要素に付加される属性として格納されます (参照された要素の子)。特定のメタデータブロックが操作されると、ブロックタグは、コマンドにより参照される要素に直接、属性として格納されます。

SET PICTURE TO LIBRARY

SET PICTURE TO LIBRARY (picture ; picRef ; picName)

引数	型		説明
picture	ピクチャー	→	新しいピクチャ
picRef	倍長整数	→	ピクチャライブラリ画像の参照番号
picName	文字	→	ピクチャの新しい名前

説明

SET PICTURE TO LIBRARY コマンドは、新規ピクチャを作成、またはピクチャライブラリにあるピクチャを置き換えます。

このコマンドを呼び出す前に、下記の引数を渡してください:

- picRefにピクチャ参照番号（1～32767の範囲）
- pictureにピクチャ自身
- picNameにピクチャの名前（最大255文字）

同じ参照番号を持つ既存のピクチャライブラリのピクチャがある場合、そのピクチャの内容は置き換えられ、引数pictureとpicNameに渡された値でピクチャと名前が変更されます。

picRefに渡された参照番号を持つピクチャライブラリのピクチャがない場合、新規ピクチャがピクチャライブラリに追加されます。

4D Server: SET PICTURE TO LIBRARYコマンドはサーバマシン上で実行される (ストアードプロシージャやトリガ) メソッドの中から使用することはできません。SET PICTURE TO LIBRARYコマンドをサーバマシン上で呼び出しても、無視され、何も行われません。

警告: デザインオブジェクト (階層リスト項目、メニュー項目等) は、ピクチャライブラリのピクチャを参照することができます。プログラムによってピクチャライブラリのピクチャを修正する際は、注意する必要があります。

Note: pictureに空のピクチャを渡すか、picRefに負数またはヌル値を渡すと、コマンドは何も行いません。

例題 1

以下の例は、ピクチャライブラリの現在の内容に関わらず、最初にユニークなピクチャ参照番号を探すことによってピクチャライブラリに新規ピクチャを追加します:

```
PICTURE LIBRARY LIST($alPicRef;$asPicNames)
Repeat
   $vlPicRef:=1+Abs(Random)
Until(Find in array($alPicRef;$vlPicRef)<0)
SET PICTURE TO LIBRARY(vgPicture;$vlPicRef;"New Picture")
```

例題 2

以下の例は、PICTURE LIBRARY LISTの3番目の例題で作成した、ディスク上のドキュメントに格納されたピクチャをピクチャライブラリの中に読み込みます:

```
SET CHANNEL(10;"")
If(OK=1)
   RECEIVE VARIABLE($vsTag)
   If($vsTag="4DV6PICTURELIBRARYEXPORT")
      RECEIVE VARIABLE($vlNbPictures)
      If($vlNbPictures>0)
         For($vlPicture;1;$vlNbPictures)
            RECEIVE VARIABLE($vlPicRef)
            If(OK=1)
               RECEIVE VARIABLE($vsPicName)
            End if
            If(OK=1)
               RECEIVE VARIABLE($vgPicture)
            End if
            If(OK=1)
               SET PICTURE TO LIBRARY($vgPicture;$vlPicRef;$vsPicName)
            Else
               $vlPicture:=$vlNbPictures+1
               ALERT("This file looks like being damaged.")
            End if
         End for
      Else
         ALERT("This file looks like being damaged.")
      End if
   Else
      ALERT("The file “"+Document+"” is not a Picture Library export file.")
   End if
   SET CHANNEL(11)
   End
```

エラー管理

ピクチャライブラリにピクチャを追加するための十分なメモリがない場合、エラーコード-108が生成されます。また、I/Oエラーが返される（例えば、ストラクチャファイルがロックされている等）点にも注意してください。エラー処理メソッドを使って、このエラーを受け取ることができます。

TRANSFORM PICTURE

TRANSFORM PICTURE (picture ; operator {; param1 {; param2 {; param3 {; param4}}}})

引数	型		説明
picture	ピクチャー	⇨	変換するソースピクチャ
		⇦	変換した結果のピクチャ
operator	倍長整数	⇨	行う変換のタイプ
param1	実数	⇨	変換パラメタ
param2	実数	⇨	変換パラメタ
param3	実数	⇨	変換パラメタ
param4	実数	⇨	変換パラメタ

説明

TRANSFORM PICTURE コマンドは、picture引数に渡したピクチャに、operatorタイプの変換を適用するために使用します。

Note: このコマンドはピクチャ変換演算子 (+/ 等, の節参照) で提供される機能を拡張します。これらの演算子は4D でも利用可能です。

コマンド実行後、ソースpictureは直接更新されます。"Crop"と"Fade to grey scale,"処理を除き処理は可逆的で、反対の処理を行うか"Reset"処理を行うことで元に戻すことができます。例えば、1%に縮小されたピクチャは後で100倍することで、変換されることなく元のサイズに戻せます。変換は元のピクチャタイプを変更しません。例えばベクタピクチャは変換後もベクタピクチャです。

operatorには実行する処理を示す数値を渡し、paramには処理に必要なパラメタを渡します。必要なパラメタの数は処理により異なります。""テーマの定数をoperatorに指定できます。処理とパラメタの説明は以下のとおりです:

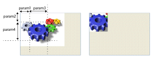
operator (値)	param1	param2	param3	param4	値
Reset (0)	-	-	-	-	
Scale (1)	幅	高さ	-	-	係数
Translate (2)	X軸	Y軸	-	-	ピクセル
Flip horizontally (3)	-	-	-	-	
Flip vertically (4)	-	-	-	-	
Crop (100)	X座標	Y座標	幅	高さ	ピクセル
Fade to grey scale (101)	-	-	-	-	

- Reset: ピクチャに対して実行されたすべてのマトリクス処理 (scale, flip等) が取り消されます。
- Scale: param1とparam2に渡された値に基づき、ピクチャは水平および垂直方向にサイズ変更されます。これらの値は係数です。例えば幅を50%拡げるにはparam1に1.5を渡します。高さを50%縮めるにはparam2に0.5を渡します。
- Translate: ピクチャは水平方向にparam1ピクセル、垂直方向にparam2ピクセル移動されます。右方向あるいは下方向に移動するには正数を、左方向あるいは上方向に移動するには負数を渡します。
- Flip horizontallyとFlip vertically: 元のピクチャは反転されます。事前に実行された移動は考慮されません。
- Crop: ピクチャは、param1とparam2座標 (ピクセル) を開始位置として、切り取られます。新しいピクチャの幅と高さをparam3とparam4で指定します。この処理は取り消すことができません。
- Fade to grey scale: ピクチャはグレースケールに変更されます。引数は必要ありません。この処理は取り消すことができません。

例題

以下はピクチャ切り取りを行う例題です (ピクチャはフォーム上で "トランケート (中央あわせなし)" フォーマットで表示されています):

```
TRANSFORM PICTURE($vpGears;Crop;50;50;100;100)
```



Original picture　Resulting picture

WRITE PICTURE FILE

WRITE PICTURE FILE (fileName ; picture {; codec})

引数	型		説明
fileName	文字	→	書き出すファイルのフルパス名, または空の文字列
picture	ピクチャー	→	書き出すピクチャフィールドまたは変数
codec	文字	→	ピクチャCodec ID

説明

WRITE PICTURE FILEコマンドを使用し、引数pictureに渡されたピクチャを、指定したcodecでディスクに保存することができます。

fileNameには、作成するファイルのフルパス名、あるいはファイル名のみを渡すことができます。ファイル名だけを渡した場合、書き出したファイルはデータベースのストラクチャファイルと同階層に置かれます。ファイルの拡張子を指定しなければなりません。
fileNameに空の文字列 ("") を渡すと、標準のファイル保存ダイアログボックスが表示され、作成するファイルの名前、場所、フォーマットをユーザが指定できます。デフォルト名がピクチャーフィールドに割り当てられている場合、ダイアログにはその名前が表示されます (SET PICTURE FILE NAMEコマンド参照)。

pictureには、ディスクに保存するピクチャを格納したピクチャ変数またはフィールドを渡します。

オプションのcodec引数を使用して、保存されるピクチャのフォーマットを指定できます。Codecは拡張子 (例 ".gif"), またはMimeタイプ (例 "image/jpeg") です。PICTURE CODEC LISTコマンドを使用して、利用可能なCodecのリストを取得できます。

codec 引数を省略すると、コマンドはfileName 引数に渡されたファイル名の拡張子に基づき、codecの決定を試みます。例えば以下のコードにおいて:

```
WRITE PICTURE FILE("c:\folder\photo.jpg";myphoto)
```

コマンドはピクチャの格納にJPEG codecを使用します。
使用された拡張子が利用可能なcodecに対応しない場合、ファイルは保存されず、OKシステム変数に0が設定されます。codec もファイル拡張子も渡さない場合、ピクチャはPICTフォーマットで保存されます。

注: (fileName の拡張子あるいはcodec 引数で指定される) picture の書き出しフォーマットが元のタイプと同じで、変換処理が適用されない場合、ピクチャはそのまま変更なしで保存されます。

コマンドが正しく実行されると、システム変数Documentには作成されたファイルのフルパス名が、システム変数OKには1が設定されます。そうでなければOKは0に設定されます。

PICTURE TYPE LIST

PICTURE TYPE LIST (formatArray {; nameArray})

引数	型		説明
formatArray	文字配列	←	読み込み/書き出しフォーマットに利用可能な QuickTimeコード
nameArray	文字配列	←	フォーマット名

互換性メモ

このコマンドは互換性の目的で保持されています。しかし、このコマンドはQuickTimeを必要とし、v11から4Dがネイティブで管理するフォーマットへのアクセスを提供していません。そのためPICTURE CODEC LISTコマンドへの置き換えをお勧めします。

QT COMPRESS PICTURE

QT COMPRESS PICTURE (picture ; method ; quality)

引数	型		説明
picture	ピクチャー	→	圧縮するピクチャ
		←	圧縮されたピクチャ
method	文字	→	4文字の圧縮方法
quality	倍長整数	→	圧縮品質 (1..1000)

互換性に関する注意

このコマンドは廃止されたメカニズムを呼び出し、また互換性のためにのみ残されています。このコマンドはCONVERT PICTURE コマンドで置き換えることができます。

説明

COMPRESS PICTURE コマンドは、pictureに渡されたフィールドまたは変数に入っているピクチャを圧縮します。

引数methodには圧縮タイプを4文字の文字列で指定します。この引数のために定数としてテーマが用意されています。

qualityには圧縮されたピクチャの品質を1から1000までの整数で指定します。一般に品質を下げるとピクチャの圧縮率が上がります。

警告: 指定された品質に対して可能な圧縮率は、圧縮するピクチャのサイズと種類によって異なります。小さなピクチャを圧縮すると、サイズが減少しないこともあります。

QT COMPRESS PICTURE FILE

QT COMPRESS PICTURE FILE (document ; method ; quality)

引数	型		説明
document	DocRef	⇨	ドキュメント参照番号
method	文字	⇨	4文字の圧縮方法
quality	倍長整数	⇨	圧縮品質 (1..1000)

互換性に関する注意

このコマンドは廃止されたメカニズムを呼び出し、また互換性のために残されています。このコマンドはWRITE PICTURE FILE または PICTURE TO BLOBコマンドで置き換えることができます。

説明

このコマンドはディスク上のピクチャドキュメントファイルを圧縮します。現在のメモリではロードできないことがわかっているピクチャを圧縮する際に、このコマンドを使用します。圧縮した後は、LOAD COMPRESS PICTURE FROM FILEコマンドを使い、メモリにロードすることができます。

Note: このコマンドはWindowsでは動作しません。

引数methodには圧縮タイプを4文字の文字列で指定します。テーマの定数を渡すことができます。

qualityには圧縮されたピクチャの品質を1から1000までの整数で指定します。一般に品質を下げるとピクチャの圧縮率が上がります。

警告: 指定された品質に対して可能な圧縮率は、圧縮するピクチャのサイズと種類によって異なります。小さなピクチャを圧縮すると、サイズが減少しないこともあります。

QT LOAD COMPRESS PICTURE FROM FILE

QT LOAD COMPRESS PICTURE FROM FILE (document ; method ; quality ; picture)

引数	型		説明
document	DocRef	→	ドキュメント参照番号
method	文字	→	4文字の圧縮方法
quality	倍長整数	→	圧縮品質 (1..1000)
picture	ピクチャー	←	圧縮されたピクチャ

互換性に関する注意

このコマンドは廃止されたメカニズムを呼び出し、また互換性のためにのみ残されています。このコマンドはREAD PICTURE FILE または CONVERT PICTUREコマンドで置き換えることができます。

説明

このコマンドはディスク上のドキュメントからロードしたピクチャを圧縮します。

Note: このコマンドはWindowsでは動作しません。

まずOpen documentを使って、PICTドキュメントを開きます。次にこの関数から返されるドキュメント参照を使い、ドキュメント中のPICTデータをロードして圧縮します。このコマンドはピクチャをメモリにロードし、指定した方法と品質に従って圧縮し、それをpictureに返します。

ピクチャは圧縮される前にメモリへロードされます。ピクチャをロードするためのメモリが足りない場合は、LOAD COMPRESS PICTURE FROM FILEコマンドを呼び出す前にCOMPRESS PICTURE FILEコマンドを使ってください。

引数methodには、圧縮タイプを4文字の文字列で指定します。テーマの定数をmethodに渡します。methodに空の文字を渡した場合、ピクチャははロードされますが、圧縮されません。

引数qualityには、圧縮されたピクチャの品質を1から1000までの整数で指定します。一般に、品質を下げると、ピクチャの圧縮率が増加します。

警告: 指定された品質に対して可能な圧縮率は、圧縮するピクチャのサイズと種類によって異なります。小さなピクチャを圧縮すると、サイズが減少しないこともあります。

例題

以下の例はファイルを開くダイアログボックスを表示し、ユーザはそこでPICTファイルを選択します。PICTファイルのピクチャがメモリにロードされて圧縮され、ピクチャ変数に格納されます。この後、ファイルが閉じられます:

```
vRef:=Open document("";"PICT")
If(OK=1)
   LOAD COMPRESS PICTURE FROM FILE(vRef;QT Photo compressor;500;vPict)
   CLOSE DOCUMENT(vRef)
End if
```

SAVE PICTURE TO FILE

SAVE PICTURE TO FILE (document ; picture)

引数	型		説明
document	DocRef	⇒	ドキュメント参照番号
picture	ピクチャー	⇒	保存するピクチャ

互換性メモ

このコマンドは廃止されたメカニズムを呼び出します。また互換性のために保持されています。このコマンドはWRITE PICTURE FILE コマンドで置き換えることができます。

説明

このコマンドは、Create documentコマンドを使って作成されたドキュメント中にpictureを保存します。

例題

以下の例はドキュメントファイルを作成してピクチャを保存します:

```
vRef:=Create document("";"PICT")
If(OK=1)
   SAVE PICTURE TO FILE(vRef;vPict)
   CLOSE DOCUMENT(vRef)
End if
```

ブール

ブールコマンド

False

Not

True

ブールコマンド

4Dにはブール演算に使用することのできる、ブール関数があります:

```
True
False
Not
```

例題

この例題ではボタンの値に基づき、ブール変数に値を設定します。myButton ボタンがクリックされたらmyBoolean にTrueを、クリックされていなければFalseを設定します。ボタンがクリックされるとボタン変数の値は1になります。

```
If(myButton=1) ` ボタンがクリックされたら
   myBoolean:=True ` myBooleanをTrueに設定
Else ` ボタンがクリックされていなければ
   myBoolean:=False ` myBooleanをFalseに設定
End if
```

上の例は以下のように一行で書くこともできます。

```
myBoolean:=(myButton=1)
```

False

False -> 戻り値

引数	型	説明
戻り値	ブール	False

説明

Falseは、ブール値のFalseを返します。

例題

この例は、変数vbOptionsにFalseを代入します:

```
vbOptions:=False
```

Not

Not (boolean) -> 戻り値

引数	型		説明
boolean	ブール	→	否定を求めるブール値
戻り値	ブール	↩	反対のブール値

説明

Not は、booleanの否定を返します。TrueをFalseに、FalseをTrueにします。

例題

以下の例は、変数にTrueを代入し変数の値をFalseにした後、再びTrueにします。

```
vResult:=True ` vResult をTrueに設定
vResult:=Not(vResult) ` vResult をFalseに設定
vResult:=Not(vResult) ` vResult をTrueに設定
```

True

True -> 戻り値

引数	型	説明
戻り値	ブール	True

説明

Trueは、ブール値のTrueを返します。

例題

この例は、変数vbOptionsにTrueを代入します:

```
vbOptions:=True
```

フォーミュラ

EDIT FORMULA

EXECUTE FORMULA

GET ALLOWED METHODS

SET ALLOWED METHODS

EDIT FORMULA

EDIT FORMULA (aTable ; formula)

引数	型		説明
aTable	テーブル	→	フォーミュラエディタにデフォルトで表示するテーブル
formula	文字変数	→	フォーミュラエディタに表示するフォーミュラを含む変 またはエディタの実を表示するには ""
		←	ユーザが確定したフォーミュラ

説明

EDIT FORMULAコマンドを使用してフォーミュラエディタを開き、ユーザはフォーミュラを作成したり変更したりできます。次の項目をデフォルトとして表示することができます。

- 左側のリストには引数aTableに渡したテーブルのフィールド。
- フォーミュラエリアには、変数formulaに格納されたフォーミュラ。formulaに空の文字列を渡した場合、デフォルトのフォーミュラは表示されず、フォーミュラエディタのみが表示されます。

ユーザは表示されたformulaを変更して保存できます。また新規にフォーミュラを記述したりロードしたりすることもできます。ユーザがダイアログを受け入れると、システム変数OKに1が設定され、formula 変数にはユーザが決定したフォーミュラが格納されます。ユーザがフォーミュラをキャンセルすると、システム変数OKに0が設定され、formula 変数は変更されません。

Note: デフォルトで、すべてのユーザーに対してメソッドとコマンドの使用は制限されます。(4D2004.4以降のバージョンにて作成されたデータベースで、DesignerとAdministratorを除く)。このメカニズムが有効であるとき、開発者はSET ALLOWED METHODSコマンドを使用して、ユーザが利用可能な要素を明示的に指定する必要があります。もしformulaが、SET ALLOWED METHODS コマンドを使用してフォーミュラエディタで許可されていないメソッドを呼び出すと、シンタックスエラーが生成され、ダイアログボックスを受け入れることはできません。

ダイアログボックスが確定したとしても、formulaが実行されないことを心に留めてください。フォーミュラの検証と変数の中身が更新されるだけです。formulaを実行する必要がある場合は、EXECUTE FORMULA コマンドを使用しなければなりません。

例題

事前に入力されたフォーミュラは使用せず、[Employees]テーブルを使用してフォーミュラエディタを表示します:

```
$myFormula:=""
EDIT FORMULA([Employees];$myFormula)
If(OK=1)
   APPLY TO SELECTION([Employees];EXECUTE FORMULA($myFormula))
End if
```

システム変数およびセット

ユーザがダイアログボックスを受け入れるとシステム変数OKに1が、キャンセルすると0が設定されます。

EXECUTE FORMULA

EXECUTE FORMULA (statement)

引数	型		説明
statement	文字	→	実行するコード

説明

EXECUTE FORMULA はstatementをコードとして実行します。ステートメントの文字列は必ず1 行だけです。statementに空の文字列を指定した場合、EXECUTE FORMULAコマンドは何も行いません。

ルールは、statementが一行のメソッドとして実行されるかぎり、それは正しく実行されます。EXECUTE FORMULA は実行速度を低下させるので、代替え手段として利用します。コンパイル済みデータベースにおいても、そのコードはコンパイルされていません。つまりstatementは実行されますが、コンパイル時にコンパイラによるチェックはされません。

statementには以下を指定できます:

- プロジェクトメソッドの呼び出し
- 4D コマンドの呼び出し
- 代入

フォーミュラにはプロセス変数とインタープロセス変数を含めることができます。しかしstatementは1行でなければならないため、(If, While, などの) フローコントロールを含めることはできません。

例題

Command name コマンドの例題を参照

GET ALLOWED METHODS

GET ALLOWED METHODS (methodsArray)

引数	型		説明
methodsArray	文字配列	←	メソッド名配列

説明

GET ALLOWED METHODS コマンドは、フォーミュラの作成に使用できるメソッド名を引数methodsArrayに返します。これらのメソッドは、エディタ上のコマンドリストの最後に一覧表示されます。

デフォルトで、メソッドはフォーミュラエディタで利用できません。メソッドはSET ALLOWED METHODS コマンドを使用して明示的に許可しなければなりません。このメソッドが実行されていない場合、GET ALLOWED METHODS メソッドは空の配列を返します。

GET ALLOWED METHODSコマンドは、SET ALLOWED METHODSコマンドに渡されたものとまったく同じ値を返します。必要に応じてコマンドは配列の作成やサイズ調整を行います。また、ワイルドカード記号(@)を使用してメソッドグループが設定されている場合、"@"記号を含む文字列が返されます(メソッドグループの名前ではありません)。

このコマンドは、特定のコンテキストのフォーミュラ(例えば、クイックレポートなど)を実行する前に、現在指定されているメソッドの組み合わせの設定を保存しておくために使用できます。

例題

この例では、レポート作成のため一連の特定メソッドを許可します:

```
  `現在の設定を取得
 GET ALLOWED METHODS(methodsArray)

  `クイックレポート用のメソッドを定義
 methodsarr_Reports{1}:="Reports_@"
 SET ALLOWED METHODS(methodsarr_Reports)
 QR REPORT([People];"MyReport")

  `先ほどの設定に戻す
 SET ALLOWED METHODS(methodsArray)
```

SET ALLOWED METHODS

SET ALLOWED METHODS (methodsArray)

引数	型		説明
methodsArray	文字配列	→	メソッド名配列

説明

SET ALLOWED METHODS コマンドを使用して、カレントセッションのフォーミュラエディタ上に表示されるメソッドを指定することができます。指 定したメソッドはコマンドリストの最後に表示され、フォーミュラで利用することができます。デフォルトでは（このコマンドを使用しない場合）、フォーミュ ラエディタ上にメソッドは表示されません。許可されていないメソッド名がフォーミュラで使用されている場合、シンタックスエラーが生成され、フォーミュラ の確定はできません。

引数methodsArrayには、フォーミュラエディタに提示するメソッドリストを格納した配列の名前を渡します。この配列は事前に定義しておかなければなりません。
メソッド名にワイルドカード記号（@）を使用し、許可されるメソッドグループを定義することができます。

デフォルトとして許可されていない4Dコマンドやプラグインコマンドをユーザが呼び出せるようにしたい場合、これらのコマンドを取り扱う特定のメソッドを使用しなくてはなりません。

Note: データベース設定の"セキュリティ"ページのオプションによって、すべてのユーザまたはDesignerとAdministratorに対し、フォーミュラエディタでのコマンドやメソッドへのアクセスを制限するメカニズムを無効にすることができるようになりました。"すべてのユーザーを制限する"オプションが チェックされていると、SET ALLOWED METHODS コマンドは効果がありません。

例題

この例は、名前が"formula"で始まるすべてのメソッドと、"Total_general"メソッドをフォーミュラエディタで使用可能にします:

```
ARRAY STRING(15;methodsArray;2)
methodsArray{1}:="formula@"
methodsArray{2}:="Total_general"
SET ALLOWED METHODS(methodsArray)
```

フォーム

- Current form name New 14.0
- FORM FIRST PAGE
- FORM Get current page
- FORM GET HORIZONTAL RESIZING
- FORM GET OBJECTS
- FORM GET PARAMETER
- FORM GET PROPERTIES
- FORM GET VERTICAL RESIZING
- FORM GOTO PAGE
- FORM LAST PAGE
- FORM LOAD Updated 14.0
- FORM NEXT PAGE
- FORM PREVIOUS PAGE
- FORM SCREENSHOT
- FORM SET HORIZONTAL RESIZING
- FORM SET INPUT
- FORM SET OUTPUT
- FORM SET SIZE
- FORM SET VERTICAL RESIZING
- FORM UNLOAD New 14.0

Current form name

Current form name -> 戻り値

引数	型		説明
戻り値	テキスト	↩	プロセス中のカレントのプロジェクトフォーム名またはカレントのテーブルフォーム名

説明

Current form nameコマンドは、プロセスのために定義されたカレントのフォームの名前を返します。カレントフォームはプロジェクトフォームでもテーブルフォームでも可能です。

FORM LOADコマンドをカレントプロセス中に使用していなければ、デフォルトでのカレントフォームは表示または印刷されているフォームです。FORM LOADコマンドをカレントプロセス中に呼び出していた場合、カレントフォームはこのコマンドによって設定されたフォームとなり、FORM UNLOAD (または CLOSE PRINTING JOB)コマンドを呼ぶまでそれが維持されます。

プロセスにおいて定義されたカレントフォームがなければ、コマンドは空の文字列を返します。

例題 1

入力フォームにおいて、以下のコードをボタンに実装します。

```
C_TEXT($FormName)
$win:=Open form window([Members];"Input";Plain form window)
DIALOG([Members];"Input")
$FormName:=Current form name
 // $FormName = "Input"
FORM LOAD([Members];"Drag")
$FormName:=Current form name
 // $FormName = "Drag"
 //...
```

例題 2

カレントフォームがプロジェクトフォームであればその名前を取得したいという場合を考えます。

```
$PointerTable:=Current form table
If(Nil($PointerTable)) // これがプロジェクトフォームであれば
   $FormName:=Current form name
   ... // 処理
End if
```

FORM FIRST PAGE

FORM FIRST PAGE
このコマンドは引数を必要としません

説明

FORM FIRST PAGE コマンドは、現在表示されているフォームページを先頭のフォームページに変更します。フォームが表示されていない場合や、すでに最初のフォームページが表示されている場合、FORM FIRST PAGEコマンドは何も行いません。

例題

以下の例は、メニューから呼び出される1行のメソッドです。これは、先頭のフォームページを表示します:

```
FORM FIRST PAGE
```

FORM Get current page

FORM Get current page {(*)} -> 戻り値

引数	型		説明
*	演算子	➡	カレントサブフォームページ番号を返す
戻り値	倍長整数	↩	現在表示されているページ番号

説明

FORM Get current pageコマンドは、現在表示されているフォームページの番号を返します。

* 引数は、複数のページを含むサブフォームタイプのページのコンテキストでコマンドが呼び出される場合に使用します。この場合、この引数を渡すと、コマンドは (コマンドを呼び出した) カレントサブフォームのページを変更します。* が省略された場合、デフォルトでコマンドは常に親フォームに適用されます。

例題

フォームにおいて、メニューバーから任意のメニューを選択、またはそのフォームが別プロセスからの呼び出しを受信した場合に、現在表示されているフォームページに応じて異なる動作を実行することができます。この例を以下に示します:

```
  ` [myTable];"myForm" フォームメソッド
 Case of
    :(Form event=On Load)
  ` ...
    :(Form event=On Unload)
  ` ...
    :(Form event=On Menu Selected)
       $vlMenuNumber:=Menu selected>>16
       $vlItemNumber:=Menu selected & 0xFFFF
       Case of
          :($vlMenuNumber=...)
             Case of
                :($vlItemNumber=...)
                :(FORM Get current page=1)
  ` ページ 1のアクション
                :(FORM Get current page=2)
  ` ページ 2のアクション
  ` ...
                :($vlItemNumber=...)
  ` ...
             End case
          :($vlMenuNumber=...)
  ` ...
       End case
    :(Form event=On Outside Call)
       Case of
          :(FORM Get current page=1)
  ` ページ 1のアクション
          :(FORM Get current page=2)
  ` ページ 2のアクション
       End case
  ` ...
 End case
```

FORM GET HORIZONTAL RESIZING

FORM GET HORIZONTAL RESIZING (resize {; minWidth {; maxWidth}})

引数	型		説明
resize	ブール	←	True: フォームを水平方向にリサイズ可 False: フォームを水平方向にリサイズ不可
minWidth	倍長整数	←	最小フォーム幅 (ピクセル)
maxWidth	倍長整数	←	最大フォーム幅 (ピクセル)

説明

FORM GET HORIZONTAL RESIZING コマンドはカレントフォームの水平サイズ変更プロパティをresize、minWidth、そしてmaxWidth変数に返します。これらのプロパティはデザインモードのフォームエディタ、またはカレントプロセス用にFORM SET HORIZONTAL RESIZING コマンドで設定されます。

FORM GET OBJECTS

FORM GET OBJECTS (objectsArray {; variablesArray {; pagesArray}} {; *})

引数	型	説明
objectsArray	文字配列	←フォームオブジェクト名
variablesArray	ポインター配列	←オブジェクトに関連付けられた 変数やフィールドへのポインタ
pagesArray	整数配列	←オブジェクトごとのページ番号
*	演算子	→渡された場合 = カレントページのみ

説明

FORM GET OBJECTSコマンドは、カレントフォームに存在する全オブジェクトのリストを配列形式で返します。このリストは、カレントフォームページのオブジェクトに限定することができます。このコマンドは、入力フォームおよび出力フォームの双方で使用することができます。

引数として渡した配列が事前に定義されていない場合、コマンドはその配列を作成し、サイズを自動的に設定します。しかし、アプリケーションをコンパイルする場合を考慮し、各配列を明示的に宣言することをお勧めします。

(フォーム内でユニークな) オブジェクト名を受け取る文字列配列をobjectsArrayに渡します。配列内でのオブジェクトの出現順序は意味を持ちません。。

コマンドにより代入される他の任意の配列は、1番目の配列との同期が取られます。

任意の引数variablesArrayにはポインタ配列を渡し、この配列にはオブジェクトに関連付けられている変数やフィールドへのポインタが格納されます。
オブジェクトに関連付けられた変数が存在しない場合、Nil ポインタが返されます。"サブフォーム"オブジェクトタイプが存在する場合、サブフォームのテーブルへのポインタが返されます。

3番目の配列（任意）pagesArrayには、フォームのページ番号が代入されます。この配列の各要素には、対応するオブジェクトのページ番号が格納されます。
継承フォームによってもたらされるオブジェクトは、カレントページのページ0に属しているものとみなされます。

任意の引数 * を使用すると、返されるオブジェクトのリストをフォームのカレントページに限定することができます。この引数を渡した場合、コマンドはカレントページ、ページ0、継承ページのオブジェクトだけを返します。このコマンドは、フォームのカレントページ上に存在するあらゆるオブジェクト (表示・非表示とも) を処理します。

FORM GET PARAMETER

FORM GET PARAMETER ({aTable ;} form ; selector ; value)

引数	型		説明
aTable	テーブル	→	フォームテーブル、または 引数が省略された場合、デフォルトテーブル
form	文字	→	フォーム名
selector	倍長整数	→	パラメタコード
value	倍長整数	←	パラメタの現在値

説明

FORM GET PARAMETER コマンドを使用すれば、aTableおよびformで指定したフォームのパラメータの現在値を取得ことができます。

調べるフォームパラメタはselectorで指定します。""テーマの定数を使用できます:

定数	型	値
NonInverted Objects	倍長整数	0

selectorにNonInverted Objectsを使用すると、valueにはWindowsのアプリケーションモードにおけるフォームの反転表示モードが返されます。フォームの反転表示は、Right-to-left 言語に対応したアプリケーションで使用されるモードです。詳細は4Dの*Design Reference*マニュアルを参照してください。

- valueが0のとき、フォームオブジェクトは反転されます。
- valueが1のとき、フォームオブジェクトは反転されません。

Windows のアプリケーションモード以外で実行された場合、コマンドは常に1を返します。

実際にフォームオブジェクトが反転されるかどうかについては、複数の要素が関係している点に留意してください。環境設定の"アプリケーションモードでオブジェクトを反転"、フォームプロパティの"オブジェクトを反転させない"、そしてデータベースを実行しているシステムが関係します。次の表には、各要素の組み合わせとFORM GET PARAMETERコマンドで返される値の関係が示されています:

環境設定:"アプリケーションモードでオブジェクトを反転" (1)	フォームプロパティ:オブジェクトを反転させない	WindowsにおけるRight-to-left表示	FORM GET PARAMETERから返される値
いいえ	O	O	1
		O	1
	O		1
			1
自動	O	O	1
		O	**0**
	O		1
			1
はい	O	O	1
		O	0
	O		1
			0

(1) この環境設定はSET DATABASE PARAMETER と Get database parameter コマンドを使用して読み書きすることもできます。

FORM GET PROPERTIES

FORM GET PROPERTIES ({aTable ;} formName ; width ; height {; numPages {; fixedWidth {; fixedHeight {; title}}}})

引数	型		説明
aTable	テーブル	⇒	フォームが属するテーブル、省略時はデフォルトテーブル
formName	文字	⇒	フォーム名
width	倍長整数	⇐	フォームの幅 (ピクセル)
height	倍長整数	⇐	フォームの高さ (ピクセル)
numPages	倍長整数	⇐	フォームのページ数
fixedWidth	ブール	⇐	True = 幅固定, False = 幅可変
fixedHeight	ブール	⇐	True = 高さ固定, False = 高さ可変
title	テキスト	⇐	フォームのウィンドウタイトル

説明

FORM GET PROPERTIES コマンドはformNameフォームのプロパティを返します。

widthおよびheight引数は、フォームの幅と高さをピクセル単位で返します。これはフォームプロパティのデフォルトウインドウサイズプロパティの値です:

- フォームサイズが**自動サイズ**の場合、幅と高さは縦横マージンを考慮に入れ、フォーム内の全オブジェクトが表示可能となるように計算されます。
- フォームサイズが**サイズ設定**の場合、幅と高さはプロパティに入力されたものになります。
- フォームサイズがオブジェクトを起点にするよう指定されている場合、幅と高さはこのオブジェクトの位置を元に計算されます。

numPages 引数0ページを除いたフォーム内のページ数を返します。

fixedWidth と fixedHeightには、フォームの幅と高さがサイズ変更可であるか (False) 、固定に設定されているか (True) を返します。

title 引数には、フォームエディターのプロパティリスト内で定義されているフォームのウインドウタイトルを返します。名前が定義されていないとtitle引数に空文字を返します。

FORM GET VERTICAL RESIZING

FORM GET VERTICAL RESIZING (resize {; minHeight {; maxHeight}})

引数	型		説明
resize	ブール	←	True: フォームを縦方向にリサイズ可 False: フォームを縦方向にリサイズ不可
minHeight	倍長整数	←	最小フォーム高さ (ピクセル)
maxHeight	倍長整数	←	最大フォーム高さ (ピクセル)

説明

FORM GET VERTICAL RESIZING コマンドは カレントフォームの垂直サイズ変更プロパティをresize、minHeight、そしてmaxHeight変数に返します。これらのプロパティはデザインモードのフォームエディタ、またはカレントプロセス用にFORM SET VERTICAL RESIZING コマンドで設定されます。

FORM GOTO PAGE

FORM GOTO PAGE (pageNumber {; *})

引数	型		説明
pageNumber	倍長整数	→	表示するフォームページ
*	演算子	→	カレントサブフォームのページを変更

説明

FORM GOTO PAGEコマンドは、現在表示されているフォームページをpageNumberで指定したフォームページに変更します。

フォームが表示されていない場合やpageNumber が現在のページである場合、**FORM GOTO PAGE**コマンドは何も行いません。pageNumberがフォームページ数よりも大きければ、最終ページを表示します。pageNumber に1未満の数が指定されると、先頭のフォームページを表示します。

* 引数は、複数のページを含むサブフォームタイプのページのコンテキストでコマンドが呼び出される場合に使用します。この場合、この引数を渡すと、コマンドは (コマンドを呼び出した) カレントサブフォームのページを変更します。* が省略された場合、デフォルトでコマンドは常に親フォームに適用されます。

フォームページ管理コマンドについて

標準アクションボタンはFORM FIRST PAGE、FORM LAST PAGE、FORM NEXT PAGE、FORM PREVIOUS PAGEそして**FORM GOTO PAGE**と同じ動作を行い、タブコントロールやドロップダウンリストなどのオブジェクトに割り当てることができます。可能な限り、コマンドの代わりに標準アクションボタンを使用してください。

ページコマンドは入力フォーム、もしくはダイアログウインドウに表示されているフォームに使用できます。出力フォームは最初のページのみを使います。フォームには、常に少なくとも最初の1ページが含まれます。フォームのページ数にかかわらず、各フォーム対するフォームメソッドは1つのみという点に留意してください。

表示されているページを確認するときはFORM Get current pageコマンドを使用してください。

注: フォームのデザインは1ページからXページについて、また0ページ(0ページに配置されたオブジェクトはすべてのページに表示される)について作業を行うことができます。フォームを使用してページコマンドを呼び出した場合、1ページからXページについての作業を行うことになります。0ページ目は自動的に表示されているページと組み合わされます。

例題

以下の例はボタンのオブジェクトメソッドです。これは指定したフォームページ3を表示します:

```
FORM GOTO PAGE(3)
```

FORM LAST PAGE

FORM LAST PAGE
このコマンドは引数を必要としません

説明

FORM LAST PAGE コマンドは、現在表示されているフォームページを最終のフォームページに変更します。フォームが表示されていない場合や、すでに最終のフォームページが表示されている場合、FORM LAST PAGEコマンドは何も行いません。

例題

以下の例は、メニューから呼び出される1行のメソッドです。これは、最終のフォームページを表示します:

```
FORM LAST PAGE
```

FORM LOAD

FORM LOAD ({aTable ;} form {; *})

引数	型		説明
aTable	テーブル	→	ロードするテーブルフォーム(省略時はプロジェクトフォームをロード)
form	文字	→	印刷のために開くプロジェクトフォームの名前、または 空の文字列を渡すと、カレントのプロジェクトフォームを閉じる
*	演算子	→	指定時、コマンドはコンポーネントから実行した場合にホストのデータベースコマンドが適応されます(それ以外の場合は無視されます)。

説明

FORM LOADコマンドを使用して印刷・データ解析用に form (プロジェクトフォームまたはテーブルフォーム) をロードできます。

データの印刷

このコマンドを使用するためには、OPEN PRINTING JOBコマンドを事前に使用して印刷ジョブが開かれている必要があります。OPEN PRINTING JOBコマンドは FORM UNLOADコマンドを暗示的に呼び出すので、このコンテキストでは FORM LOADコマンドを呼び出す必要があります。一度ロードされると、この form はカレントの印刷フォームとなります。Print object コマンドを含む、全てのオブジェクト管理コマンドはこのフォームに対して動作します。

もしも印刷フォームが事前(**FORM LOAD** コマンドを呼び出す前)にロードされていた場合は、そのフォームは閉じられ、formにより置き換えられます。ひとつの印刷セッションで複数のプロジェクトフォームを開いたり閉じたりできます。**FORM LOAD**で印刷フォームを変更してもページブレークは生成されません。ページブレークを管理するのは開発者の仕事です。

プロジェクトフォーム(またはフォームのオブジェクトメソッド)を開く際は、On Load フォームイベントのみが実行されます。他のフォームイベントは無視されます。印刷の終わりには On Unload フォームイベントが実行されます。

フォームのグラフィックな一貫性を保持するために、プラットフォームにかかわらず"印刷"アピアランスプロパティを適用することをお勧めします。

CLOSE PRINTING JOBコマンドが呼び出されると、カレント印刷フォームは自動で閉じられます。

Compatibility note: v14以前のバージョンの4Dでは、**FORM LOAD**コマンド(v14以前はOPEN PRINTING FORMという名称)は form引数に空の文字列を渡す事によってカレントのプロジェクトフォームを閉じていました。このシンタックスはv14以降サポートされておらず、渡してもエラーが返されます。フォームを閉じる際には FORM UNLOAD コマンドまたは CLOSE PRINTING JOBを使用して下さい。

コンテンツの解析

ここではデータ解析のためスクリーン外のフォームをロードする使用法を解説します。これをするためには、**FORM LOAD** を印刷ジョブ以外のコンテキストで呼び出すだけです。この際にはフォームイベントは実行されません。

FORM LOAD はフォームコンテンツを処理するために FORM GET OBJECTS コマンドと OBJECT Get typeコマンドと合わせて使用することができます。その後、フォームをメモリーから解放するために FORM UNLOADコマンドを呼び出す必要があります。

どの場合においても、スクリーン上のフォームはロードされたままである(**FORM LOAD** コマンドによって影響されない)ので、FORM UNLOADコマンドを呼び出したあとリロードする必要はないということです。

コマンドがコンポーネントから実行された場合、デフォルトでコンポーネントフォームを呼び出します。* 演算子を渡した場合、メソッドはホストデータベースフォームをロードします。

例題 1

印刷ジョブにプロジェクトフォームを呼び出す場合:

```
OPEN PRINTING JOB
FORM LOAD("print_form")
 // イベントとオブジェクトメソッドの実行
```

例題 2

印刷ジョブにテーブルフォームを呼び出す場合:

```
OPEN PRINTING JOB
FORM LOAD([People];"print_form")
 // イベントとオブジェクトメソッドの実行
```

例題 3

フォームの内容を解析してテキスト入力エリアに何らかの処理をする場合:

```
FORM LOAD([People];"my_form")
 // イベントやメソッドを実行することなくフォームを選択
FORM GET OBJECTS(arrObjNames;arrObjPtrs;arrPages;*)
For($i;1;Size of array(arrObjNames))
   If(OBJECT Get type(*;arrObjNames{$i})=Object type text input)
 //… processing
   End if
End for
FORM UNLOAD
```

FORM NEXT PAGE

FORM NEXT PAGE
このコマンドは引数を必要としません

説明

FORM NEXT PAGE コマンドは、現在表示されているフォームページから次のフォームページに移動します。フォームが表示されていない場合や、すでに最終のフォームページが表示されている場合、FORM NEXT PAGE コマンドは何も行いません。

例題

以下の例は、メニューから呼び出される1行のメソッドです。これは、現在の次のフォームページを表示します:

```
FORM NEXT PAGE
```

FORM PREVIOUS PAGE

FORM PREVIOUS PAGE
このコマンドは引数を必要としません

説明

FORM PREVIOUS PAGE コマンドは、現在表示されているフォームページから前のフォームページに移動します。フォームが表示されていない場合や、すでに先頭のフォームページが表示されている場合、FORM PREVIOUS PAGE コマンドは何も行いません。

例題

以下の例は、メニューから呼び出される1行のメソッドです。これは、前のフォームページを表示します:

```
FORM PREVIOUS PAGE
```

FORM SCREENSHOT

FORM SCREENSHOT ({{aTable ;} formName ;} formPict {; pageNum})

引数	型		説明
aTable	テーブル	⇒	フォームテーブル
formName	テキスト	⇒	フォーム名
formPict	ピクチャー	⇐	第一引数が省略された場合実行中のフォームのピクチャー。フォーム名がが渡された場合フォームエディター中のフォームのピクチャー
pageNum	倍長整数	⇒	フォームページ番号

説明

FORM SCREENSHOT コマンドはフォームをピクチャーにして返します。このコマンドは2つのシンタックスを受け入れます。使用されるシンタックスに応じてフォームのピクチャー、またはフォームエディター内のフォームのピクチャーが返されます。

- FORM SCREENSHOT (*formPict*)
  このシンタックスでは実行中、または FORM LOAD コマンドによってロード中のフォームのカレントページのスクリーンショットが返されます。formPict引数に返されるピクチャーにはフォームのすべての表示可能なオブジェクトが含まれ、そこにカレントのフィールドや変数の値が表示されます。フォームのすべてが返され、ウィンドウサイズは考慮されません。
  このシンタックスは入力フォームでのみ有効です。

- FORM SCREENSHOT ({*aTable* ;} *formName* ; *formPict* {; *pageNum*})
  このシンタックスはフォームエディターに表示されるフォームテンプレートのスクリーンショットを返します。すべての表示可能なオブジェクトはフォームエディターに表示される通りに描画されます。継承フォームや0ページのオブジェクトも含まれます。
  テーブルフォームのスクリーンショットを得たい場合、aTable引数にフォームテーブルを渡し、formNameにフォーム名を渡します。プロジェクトフォームの場合、formNameにフォーム名を渡します。
  デフォルトでこのコマンドはフォームの1ページ目のスクリーンショットを返します。0ページやその他のページのピクチャーを得たい場合、pageNum引数を使用してページ番号を指定します。

FORM SET HORIZONTAL RESIZING

FORM SET HORIZONTAL RESIZING (resize {; minWidth {; maxWidth}})

引数	型		説明
resize	ブール	→	True: フォームを横方向にリサイズ可能 False: フォームを横方向にリサイズ不可
minWidth	倍長整数	→	最小幅 (ピクセル)
maxWidth	倍長整数	→	最大幅 (ピクセル)

説明

FORM SET HORIZONTAL RESIZING コマンドを使用すると、プログラムからカレントフォームの水平リサイズプロパティを変更することができます。デフォルトとして、これらのプロパティはデザインモードのフォームエディタにおいて設定することができます。新しいプロパティはカレントプロセスに対して設定され、フォームと一緒に保存されません。

resize 引数を用いて、フォームを水平方向にリサイズできるかどうか、つまり、幅の変更が可能かどうかを定義します (ユーザが手動で、またはプログラムから変更)。
Trueを渡すと、ユーザはフォームの幅を変更することができます。4DはマーカとしてminWidthとmaxWidthに代入された値を使用します。
Falseを渡すと、ユーザはカレントフォームの幅を変更できません。この場合、minWidthとmaxWidthに値を渡す必要はありません。

1番目の引数にTrueを渡した場合は、任意の引数minWidthとmaxWidthに最小幅と最大幅の新しい値（ピクセル単位）を渡すことができます。これらの引数を省略した場合、デザインモードで設定した値（設定されている場合）が使用されます。

例題

FORM SET SIZE コマンドを参照してください。

FORM SET INPUT

FORM SET INPUT ({aTable ;} form {; userForm}{; *})

aTable	テーブル	⇒	入力フォームを設定するテーブル, または 省略した場合、デフォルトテーブル
form	文字	⇒	入力フォームとして設定するフォーム名
userForm	文字	⇒	使用するユーザフォーム名
*		⇒	自動ウィンドウサイズ

説明

FORM SET INPUTコマンドは、aTableのカレント入力フォームをformまたはuserFormに設定します。フォームはaTableに属していなければなりません。

このコマンドのスコープは、カレントプロセスです。各テーブルは、プロセスごとに個々の入力フォームを持っています。

Note: 構造的な理由から、このコマンドはプロジェクトフォームと互換がありません。プロジェクトフォームをformに渡しても、コマンドは何も行いません。

FORM SET INPUT はフォームを表示しません。データ入力や読み込み、他のコマンドで使用するフォームを指定するだけです。フォームの作成に関する詳細は*4D Design Reference*マニュアルを参照してください。

デフォルト入力フォームはテーブルごとにエクスプローラウィンドウで指定します。このデフォルト入力フォームは、FORM SET INPUTコマンドで入力フォームを指定しない場合や、指定したフォームが存在しない場合に使用されます。

任意の引数userForm を使用して、(formを基に作成された) ユーザフォームをデフォルトの入力フォームとして設定することができます。有効なユーザフォーム名を渡すと、カレントプロセスの入力フォームに代わり、このフォームがデフォルトとして使用されます。これにより、(CREATE USER FORMコマンドで生成された) 複数の異なるカスタムユーザフォームを同時に使用できるようになり、現在の状況で適切なフォームを使用することができます。
ユーザフォームに関する詳細はの節を参照してください。

入力フォームは多くのコマンドで表示されますが、一般にデータの入力や修正に使用されます。以下のコマンドは、データ入力や検索用に入力フォームを表示します:

- ADD RECORD
- DISPLAY RECORD
- MODIFY RECORD
- QUERY BY EXAMPLE

DISPLAY SELECTION や MODIFY SELECTIONコマンドは、出力フォームを使用してレコードのリストを表示します。ユーザがリスト上のレコードをダブルクリックすると、入力フォームを表示します。

データ読み込みコマンドIMPORT TEXT, IMPORT SYLK そして IMPORT DIFは、レコードの読み込みにカレント入力フォームを使用します。

オプション引数 * は、デザインモードのフォームプロパティウィンドウおよびOpen windowで使用されます。* を指定することにより、(ダイアログボックスや入力フォームとして) 次回フォームを使用する際、フォームプロパティの設定をもとに自動的にウィンドウサイズを変更するよう4Dに指示します。詳しくはOpen windowの節を参照してください。

Note: オプション引数 * を使用するしないに関係なく、FORM SET INPUTコマンドはテーブルの入力フォームを変更します。

例題 1

以下の例は、FORM SET INPUTコマンドの一般的な使用方法です:

```
FORM SET INPUT([Companies];"New Comp") ` 新しい会社を追加するフォーム
ADD RECORD([Companies]) ` 新しい会社の追加
```

例題 2

複数の会社を処理する請求書データベースでは、対応するユーザフォームを用いて請求書を作成しなくてはなりません:

```
Case of
   :(company="4D SAS")
      FORM SET INPUT([Invoices];"Input";"4D_SAS")
   :(company="4D Inc")
      FORM SET INPUT([Invoices];"Input";"4D_Inc")
   :(company="Acme")
      FORM SET INPUT([Invoices];"Input";"ACME")
End case
ADD RECORD([Factures])
```

FORM SET OUTPUT

FORM SET OUTPUT ({aTable ;} form {; userForm})

引数	型	説明
aTable	テーブル	→ 出力フォームを設定するテーブル, または 省略した場合、デフォルトテーブル
form	文字	→ フォーム名
userForm	文字	→ 使用するユーザフォーム名

説明

FORM SET OUTPUTコマンドは、formまたはuserFormをaTableのカレント出力フォームとして設定します。このフォームはaTableに属するものでなければなりません。

このコマンドのスコープはカレントプロセスです。各テーブルはプロセスごとに出力フォームを持っています。

Note: 構造的な理由から、このコマンドはプロジェクトフォームと互換がありません。プロジェクトフォームをformに渡しても、コマンドは何も行いません。

FORM SET OUTPUT はフォームを表示しません。データ印刷や表示、他のコマンドで使用するフォームを指定するだけです。フォームの作成に関する詳細は*4D Design Reference*マニュアルを参照してください。

デフォルト出力フォームはテーブルごとにエクスプローラウィンドウで指定します。このデフォルト出力フォームは、FORM SET OUTPUTコマンドで出力フォームを指定しない場合や、指定したフォームが存在しない場合に使用されます。

任意の引数userForm を使用して、(formを基に作成された) ユーザフォームをデフォルトの出力フォームとして設定することができます。有効なユーザフォーム名を渡すと、カレントプロセスの出力フォームに代わり、このフォームがデフォルトとして使用されます。これにより、(CREATE USER FORMコマンドで生成された) 複数の異なるカスタムユーザフォームを同時に使用できるようになり、現在の状況で適切なフォームを使用することができます。
ユーザフォームに関する詳細はの節を参照してください。

出力フォームは3つのコマンドグループ(画面上にレコードをリスト表示するグループ、レポートを作成するグループ、データを書き出すグループ)で使用されます。DISPLAY SELECTIONやMODIFY SELECTIONコマンドは、出力フォームを使用してレコードのリストを表示します。PRINT LABELやPRINT SELECTIONコマンドを使用してレポートを作成する際にも出力フォームを使用します。各データ書き出しコマンド(EXPORT DIF, EXPORT SYLK そして EXPORT TEXT)でも出力フォームを使用します。

例題

以下の例は、FORM SET OUTPUTコマンドの典型的な使用方法です。この例ではFORM SET OUTPUTコマンドを出力フォームが使用される直前に記述していますが、直前である必要はありません。実際、このFORM SET OUTPUTコマンドがこのメソッドの前に同じプロセス内で実行されていれば、このコマンドを全く別のメソッドで実行しても構いません:

```
FORM SET INPUT([Parts];"Parts In") ` 入力フォームを設定
FORM SET OUTPUT([Parts];"Parts List") ` 出力フォームを設定
MODIFY SELECTION([Parts]) ` このコマンドは両フォームを使用する
```

FORM SET SIZE

FORM SET SIZE ({object ;} horizontal ; vertical {; *})

引数	型		説明
object	文字	→	フォームの境界を指定するオブジェクト名
horizontal	倍長整数	→	* が渡されていれば: 水平マージン (ピクセル) * が省略されていれば: 幅 (ピクセル)
vertical	倍長整数	→	* が渡されていれば: 垂直マージン (ピクセル) * が省略されていれば: 高さ (ピクセル)
*	演算子	→	渡されれば: horizontalとvertical引数で 指定されたマージンを加える (自動サイズまたはobjectが指定されればそれを基とととしたサイズ) 省略すれば: horizontalとverticalをフォームの幅と高さにする

説明

FORM SET SIZE コマンドを使用すると、プログラムからカレントフォームのサイズを変更することができます。新しいサイズは、カレントプロセスに対して定義され、フォームには保存されません。

デザインモードと同様に、このコマンドを使用して、3通りの方法でフォームサイズを設定することができます:

- 自動: すべてのオブジェクトが表示されるよう、4Dがフォームサイズを決定。縦横マージンを追加可能。
- 指定したフォームオブジェクトの位置に基づき決定する。縦横マージンを追加可能。
- 幅と高さを直接指定。

フォームのリサイズに関する詳細は、*4D Design Reference*マニュアルを参照してください。

自動サイズ
自動でフォームサイズを設定したい場合、以下のシンタックスを使用します:

```
FORM SET SIZE(horizontal;vertical;*)
```

この場合、horizontal と verticalにはそれぞれ右と下に追加するマージンをピクセル単位で渡します。

オブジェクトを基にしたサイズ
オブジェクトを基にフォームサイズを決定したい場合、以下のシンタックスを使用します:

```
FORM SET SIZE(object;horizontal;vertical)
```

この場合、horizontal と verticalにはそれぞれオブジェクトの右と下に追加するマージンをピクセル単位で渡します。* 引数を渡すことはできません。

サイズ指定
フォームサイズを固定で指定したい場合、下のシンタックスを使用します:

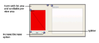
```
FORM SET SIZE(horizontal;vertical)
```

この場合、horizontal と verticalにはそれぞれフォームの高さと幅を指定します。

FORM SET SIZEコマンドはフォームサイズを変更しますが、サイズ調整プロパティも考慮します。例えば、フォームの最小幅が500ピクセルの場合に、コマンドで幅を400ピクセルに設定すると、新しいフォーム幅は500ピクセルになります。
また、このコマンドはフォームウィンドウのサイズは変更しないという点に注意してください。ウインドウサイズを変更しないでフォームサイズを変えることもできます(その逆も同様)。フォームウィンドウのサイズを変更する方法については、RESIZE FORM WINDOWコマンドを参照してください。

例題

次の例題は、エクスプローラタイプのウィンドウの設定方法を示しています。以下のフォームはデザインモードで作成します:

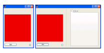


フォームのサイズは "自動"です。

ウィンドウは以下のコードで表示されます:

```
Sref:=Open form window([Table1];"Form1";Plain form window;Horizontally Centered;Vertically Centered;*)
DIALOG([Table1];"Form1")
CLOSE WINDOW
```

ウィンドウの右の部分はオプションのクリックにより表示されたり隠されたりします。:



このボタンに関連付けられたメソッドは以下のようになります:

```
Case of
   :(Form event=On Load)
      C_BOOLEAN(bl;<>collapsed)
      C_LONGINT(margin)
      margin:=15
      bl:=<>collapsed
      If(<>collapsed)
         FORM SET HORIZONTAL RESIZING(False)
         FORM SET SIZE("bl";margin;margin)
      Else
         FORM SET HORIZONTAL RESIZING(True)
         FORM SET SIZE("tab";margin;margin)
      End if

   :(Form event=On click)
      <>collapsed:=bl
      If(bl)
 `collapsed
         OBJECT GET COORDINATES(*;"bl";$l;$t;$r;$b)
         GET WINDOW RECT($lf;$tf;$rf;$bf;Current form window)
         SET WINDOW RECT($lf;$tf;$lf+$r+margin;$tf+$b+margin;Current form window)
         FORM SET HORIZONTAL RESIZING(False)
         FORM SET SIZE("bl";margin;margin)

      Else
 `expanded
         OBJECT GET COORDINATES(*;"tab";$l;$t;$r;$b)
         GET WINDOW RECT($lf;$tf;$rf;$bf;Current form window)
         SET WINDOW RECT($lf;$tf;$lf+$r+margin;$tf+$b+margin;Current form window)
         FORM SET HORIZONTAL RESIZING(True)
         FORM SET SIZE("tab";margin;margin)
      End if

End case
```

FORM SET VERTICAL RESIZING

FORM SET VERTICAL RESIZING (resize {; minHeight {; maxHeight}})

引数	型		説明
resize	ブール	→	True: フォームを縦方向にリサイズ可 False: フォームを縦方向にリサイズ不可
minHeight	倍長整数	→	最小高さ (ピクセル)
maxHeight	倍長整数	→	最大高さ (ピクセル)

説明

FORM SET VERTICAL RESIZINGコマンドを使用すると、プログラムからカレントフォームの垂直リサイズプロパティを変更することができます。デフォルトで、これらのプロパティはデザインモードのフォームエディタにおいて設定することができます。新しいプロパティはカレントプロセスに対して設定され、フォームと一緒に保存されません。

resize 引数を用いて、フォームを水平方向にリサイズできるかどうか、つまり、高さの変更が可能かどうかを定義します (ユーザが手動で、またはプログラムから変更)。
Trueを渡すと、ユーザはフォームの高さを変更することができます。4DはマーカとしてminHeightとmaxHeightに代入された値を使用します。
Falseを渡すと、ユーザはカレントフォームの高さを変更できません。この場合、minHeightとmaxHeightに値を渡す必要はありません。

1番目の引数にTrueを渡した場合は、任意の引数minHeightとmaxHeightに最小高さと最大高さの新しい値（ピクセル単位）を渡すことができます。
これらの引数を省略した場合、デザインモードで設定した値（設定されている場合）が使用されます。

例題

FORM SET SIZE コマンドの例題参照

FORM UNLOAD

FORM UNLOAD
このコマンドは引数を必要としません

説明

FORM UNLOADコマンドは、FORM LOADコマンドで指定したカレントのフォームをメモリーから解放します。.

このコマンドは FORM LOADコマンドを印刷以外の目的で使用したときには必ず呼び出さなければなりません(実際に印刷をしたときには、CLOSE PRINTING JOB コマンドを呼び出した際に自動的にカレントフォームは再度閉じられます)。

フォームイベント

- CALL SUBFORM CONTAINER
- Contextual click
- Form event Updated 14.0
- Right click
- SET TIMER
- Activated
- After
- Before
- Deactivated
- During
- In break
- In footer
- In header
- Outside call

CALL SUBFORM CONTAINER

CALL SUBFORM CONTAINER (event)

引数	型		説明
event	倍長整数	→	送信するイベント

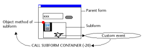
説明

CALL SUBFORM CONTAINER コマンドを使用してサブフォームインスタンスからそれを含むサブフォームコンテナにイベントを送信できます。そしてサブフォームコンテナは親フォームのコンテキストでイベントを処理できます。

このコマンドはサブフォームのフォームメソッドまたはサブフォーム上のオブジェクトのオブジェクトメソッドに置かれなければなりません。サブフォームコンテナのオブジェクトメソッドだけがイベントを受信します。

eventには4Dの定義済みフォームイベント ("Form Events"テーマの定数を使用できます) 、あるいはカスタムイベントに対応する値をを渡すことができます。前者の場合、イベントはサブフォームに対して有効にされていなければなりません。カスタムイベントの場合、既存あるいは将来の4Dのイベント番号との重複を避けるため、eventに負数を渡すことを推奨します。

CALL SUBFORM CONTAINER *コマンドの実行例:*


Parent form
xxx
Object method of subform
Subform
Custom event
CALL SUBFORM CONTAINER (-20)

Contextual click

Contextual click -> 戻り値

引数	型		説明
戻り値	ブール	←	コンテキストクリックを検知した場合True、そうでなければFalse

説明

Contextual click コマンドは、コンテキストクリックが行われた場合にTrueを返します。

- WindowsとMac OSにおいて、コンテキストクリックはマウスの右ボタンを使用して行います。
- Mac OSにおいて、コンテキストクリックは、**Control+クリック**でも行うことができます。

このコマンドはOn clickedフォームイベントのコンテキストで使用します。したがって、デザインモードにおいて、このイベントがフォームや特定のオブジェクトのプロパティで適切に選択されていることを確認する必要があります。

例題

このメソッドをスクロールエリアと組み合わせて使用すると、コンテキストメニューを用いて配列要素の値を変更することができます:

```
If(Contextual click)
   If(Pop up menu("True;False")=1)
      myArray{myArray}:="True"
   Else
      myArray{myArray}:="False"
   End if
End if
```

Form event

Form event -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	フォームイベント番号

説明

Form eventコマンドは、現在生成中のフォームイベントタイプを示す数値を返します。通常フォームやオブジェクトメソッド内で**Form event**を使用します。

4DにはForm Eventsテーマで定義された定数が用意されており、**Form event**から返される値と比較することができます。

イベントには、一般的なイベント(任意のタイプのオブジェクトに対して生成される)と、特定タイプのオブジェクトのみに発生するイベントがあります。

定数	型	値	コメント
On Load	倍長整数	1	フォームが表示または印刷されようとしている
On Validate	倍長整数	3	レコードのデータ入力が受け入れられた
On Clicked	倍長整数	4	オブジェクト上でクリックされた
On Header	倍長整数	5	フォームのヘッダエリアが印刷あるいは表示されようとしている
On Printing Break	倍長整数	6	フォームのブレークエリアのひとつが印刷されようとしている
On Printing Footer	倍長整数	7	フォームのフッタエリアが印刷されようとしている
On Display Detail	倍長整数	8	レコードがリスト中に、あるいは行がリストボックス中に表示されようとしている
On Outside Call	倍長整数	10	フォームがCALL PROCESSによる呼び出しを受けた
On Activate	倍長整数	11	フォームウィンドウが最前面のウィンドウになった
On Deactivate	倍長整数	12	フォームウィンドウが最前面のウィンドウでなくなった
On Double Clicked	倍長整数	13	オブジェクト上でダブルクリックされた
On Losing Focus	倍長整数	14	フォームオブジェクトがフォーカスを失った
On Getting Focus	倍長整数	15	フォームオブジェクトがフォーカスを得た
On Drop	倍長整数	16	データがオブジェクトにドロップされた
On Before Keystroke	倍長整数	17	フォーカスのあるオブジェクトに文字が入力されようとしている。Get edited textはこの文字を含まないオブジェクトのテキストを返す
On Menu Selected	倍長整数	18	メニュー項目が選択された
On Plug in Area	倍長整数	19	外部オブジェクトのオブジェクトメソッドの実行がリクエストされた
On Data Change	倍長整数	20	オブジェクトのデータが変更された
On Drag Over	倍長整数	21	データがオブジェクト上にドロップされる可能性がある
On Close Box	倍長整数	22	ウィンドウのクローズボックスがクリックされた
On Printing Detail	倍長整数	23	フォームの詳細エリアが印刷されようとしている
On Unload	倍長整数	24	フォームを閉じる、あるいは解放しようとしている
On Open Detail	倍長整数	25	レコードがダブルクリックされて、入力フォームを開こうとしている
On Close Detail	倍長整数	26	入力フォームから離れ、出力フォームに移動しようとしている
On Timer	倍長整数	27	SET TIMERコマンドで設定した時間が経過した
On After Keystroke	倍長整数	28	フォーカスのあるオブジェクトに文字が入力されようとしている。Get edited textはこの文字を含むオブジェクトのテキストを返す
On Resize	倍長整数	29	フォームウィンドウがリサイズされた
On After Sort	倍長整数	30	(リストボックスのみ) リストボックスの列中で標準のソートが行われた
On Selection Change	倍長整数	31	• リストボックス: 現在の行や列の選択が変更された • リスト中のレコード: リストフォームまたはサブフォームにおいて、カレントレコードあるいはカレントセレクションの行選択が変更された • 階層リスト: リスト中の選択がクリックやキーストロークなどで変更された • 入力可フィールドや変数 (v12.x以降): クリックやキー押下により、選択されたテキストやカーソルの位置がエリア内で変更された
On Column Moved	倍長整数	32	(リストボックスのみ) リストボックスの列がユーザのドラッグ＆ドロップで移動された
On Column Resize	倍長整数	33	(リストボックスのみ) リストボックスの列幅が変更された
On Row Moved	倍長整数	34	(リストボックスのみ) リストボックスの行がユーザのドラッグ＆ドロップで移動された
On Mouse Enter	倍長整数	35	マウスカーソルがオブジェクトの描画エリア内に入った
On Mouse Leave	倍長整数	36	マウスカーソルがオブジェクトの描画エリアから出た
On Mouse Move	倍長整数	37	マウスカーソルがオブジェクトの描画エリア上で (最低1ピクセル) 動いたか、変更キー(Shift, Alt, Shift Lock)が押された イベントがオブジェクトに対してのみチェックされていた場合は、マウスカーソルがオブジェクトのグラフィックエリアの中にあった場合にのみイベントが生成されます。
On Arrow Click	倍長整数	38	(3Dボタンのみ)3D ボタンの"三角"エリアがクリックされた
On Long Click	倍長整数	39	(3Dボタンのみ) 3D ボタンがクリックされ、特定の時間以上マウスボタンが押され続けている
On Load Record	倍長整数	40	リスト更新中、更新中にレコードがロードされた (ユーザがレコード行をクリックし、フィールドが編集モードになった)
On Before Data Entry	倍長整数	41	(リストボックスのみ) リストボックスセルが編集モードに変更されようとしている
On Header Click	倍長整数	42	(リストボックスのみ) リストボックスの列ヘッダでクリックが行われた
On Expand	倍長整数	43	(階層リストまたは階層リストボックスのみ) クリックやキーストロークで階層リストの要素が展開された
On Collapse	倍長整数	44	(階層リストまたは階層リストボックスのみ) クリックやキーストロークで階層リストの要素が折りたたまれた
On After Edit	倍長整数	45	フォーカスのあるオブジェクトの内容が更新された
On Begin Drag Over	倍長整数	46	オブジェクトがドラッグされている
On Begin URL Loading	倍長整数	47	(Webエリアのみ) 新しいURLがWeb エリアにロードされた
On URL Resource Loading	倍長整数	48	(Webエリアのみ) 新しいリソースがWeb エリアにロードされた
On End URL Loading	倍長整数	49	(Webエリアのみ) URLのすべてのリソースがロードされた
On URL Loading Error	倍長整数	50	(Webエリアのみ) URLをロード中にエラーが発生した
On URL Filtering	倍長整数	51	(Webエリアのみ) Web エリアがURLをブロックした
On Open External Link	倍長整数	52	(Webエリアのみ) 外部URLがブラウザで開かれた
On Window Opening Denied	倍長整数	53	(Webエリアのみ) ポップアップウィンドウがブロックされた
On bound variable change	倍長整数	54	サブフォームにバインドされた変数が更新された
On Mac toolbar button	倍長整数	55	Mac OSにおいて、ユーザーがツールバー管理ボタンをクリックした
On Page Change	倍長整数	56	フォーム中のカレントページが変更された
On Footer Click	倍長整数	57	(リストボックスのみ) リストボックスあるいはリストボックス列でフッターがクリックされた
On Delete Action	倍長整数	58	(階層リストとリストボックスのみ) ユーザーが項目の削除を試みた
On Picture Scroll	倍長整数	59	マウスやキーボードを使用して、ユーザーがピクチャーフィールドや変数の内容をスクロールした。

注: 出力フォーム用のイベントを**プロジェクトフォーム**で実装することはできません。関連するイベントは以下の通りです: On Display Detail, On Open Detail, On Close Detail, On Load Record, On Header, On Printing Detail, On Printing Break, On Printing Footer

イベントとメソッド

フォームイベントが発生すると、4Dは以下のアクションを行います:

- まず4Dはフォーム中のオブジェクトをブラウズし、発生したオブジェクトイベントがプロパティで選択されているすべてのオブジェクトのオブジェクトメソッドを呼び出します。
- 次に、発生したイベントに対応するフォームイベントがプロパティで選択されていれば、フォームメソッドを呼び出します。

オブジェクトメソッドが特定の順序で呼び出されることを期待することはできません。おおざっぱに言って、オブジェクトメソッドは常にフォームメソッドよりも前に呼び出されます。オブジェクトがサブフォームの場合、サブフォームリストフォームのオブジェクトメソッドが呼び出され、次にリストフォームのフォーム メソッドが呼び出されます。そして4Dは引き続き、親フォームのオブジェクトメソッドを呼び出します。言い換えれば、オブジェクトがサブフォームの時、4Dはサブフォームオブジェクト内で、オブジェクトとフォームメソッドの関係と同じルールを適用します。

On Load と On Unload イベントを除き、発生したイベントがフォームイベントプロパティで選択されていなかったとしても、オブジェクトプロパティで選択されていればそのオブジェクトメソッドの呼び出しが妨げられることはありません。言い換えれば、フォームレベルでイベントを有効あるいは無効にしても、オブジェクトイベントプロパティには影響ありません。

特定のイベントに関連するオブジェクトの数は、イベントの性質により異なります:

- On Load イベント - On Loadオブジェクトイベントプロパティが選択されている、フォームのすべてのページのすべてのオブジェクトのオブジェクトメソッドが呼び出されます。そしてOn Loadフォームイベントプロパティが選択されていれば、フォームメソッドが呼び出されます。
- On Activate や On Resize イベント - これらのイベントは個々のオブジェクトには適用されず、フォームに適用されるため、オブジェクトメソッドは呼び出されません。ゆえにOn Activate フォームイベントプロパティが選択されてれば、そのフォームメソッドのみが呼び出されます。
- On Timer イベント - このイベントは事前にSET TIMERコマンドが使用された場合にのみ生成されます。On Timer フォームイベントプロパティが選択されていると、フォームメソッドのみがイベントを受け取ります。オブジェクトメソッドは呼び出されません。
- On Drag Over イベント - "ドロップ可"プロパティが選択されていれば、イベント中で関連するドロップ可能なオブジェクトのみオブジェクトメソッドが呼び出されます。フォームメソッドは呼び出されません。
  逆にOn Begin Drag Overイベントについては、ドラッグされているオブジェクトのオブジェクトメソッドやフォームメソッドが呼び出されます ("ドラッグ可"プロパティが選択されていれば)。

警告: 他のすべてのイベントと異なり、On Begin Drag Over やイベント中、呼び出されるメソッドは、ドラッグ＆ドロップソースオブジェクトのプロセスのコンテキストで実行されます。ドラッグ＆ドロップ先のオブジェクトではありません。詳細はドラッグ＆ドロップを参照してください。

- フォームのOn Mouse Enter, On Mouse Move および On Mouse Leave イベントが選択されていると、これらのイベントはフォームオブジェクトごとに生成されます。これらがオブジェクトで有効にされている場合、イベントはこの オブジェクトに対してのみ生成されます。多層構造のオブジェクトの場合、上位レベルから下位レベルに向けてそのイベントを処理できるオブジェクトを探し、最初に見つかったオブジェクトによりイベントが生成されます。OBJECT SET VISIBLE コマンドを使用して非表示にされたオブジェクトでは、これらのイベントは生成されません。オブジェクト入力中、他のオブジェクトはマウスの位置によりこの タイプのイベントを受け取るかもしれません。
  On Mouse Move イベントはマウスカーソルが動いたときだけではなく、ユーザーが変更キー(**Shift**, **Shift Lock**, **Ctrl** または **Option**)を押したときにも発生することに注意して下さい(これにより、ドラッグ&ドロップによるコピーや移動も管理できるようになります)。

- リスト中のレコード: DISPLAY SELECTION / MODIFY SELECTIONで表示されるリストフォームやサブフォームでメソッドやフォームイベントが呼び出される順序は以下のとおりです:
  ヘッダーエリアのそれぞれのオブジェクトごとに:
    - オブジェクトメソッドのOn Header イベント
    - フォームメソッドのOn Header イベント

  レコードごとに:
    - 詳細エリアのオブジェクトごとに:
      - オブジェクトメソッドのOn Display Detail イベント
    - フォームメソッドのOn Display Detail イベント

- On Display Detail や On Headerイ ベントでダイアログボックスを表示する4Dコマンドを呼び出すことはできません。これはシンタックスエラーを起こします。以下のコマンドが関連します: ALERT, DIALOG, CONFIRM, Request, ADD RECORD, MODIFY RECORD, DISPLAY SELECTION そして MODIFY SELECTION。
- On Page Change: このイベントはフォームレベルでのみ利用でき (フォームメソッド内でのみ使用します)、フォームのカレントページが変更されるたびに生成されます (FORM GOTO PAGE コマンドの呼び出しや標準ナビゲーションアクションに伴い)。ページが完全にロードされた後に呼び出されることに留意してください。 例えば (Webエリアを含む) すべてのオブジェクトが初期化された後です。このイベントはすべてのオブジェクトが初期化済みの状態で実行する必要のあるコードがあるときに有用です。また フォームがロードされたときにすべてのコードを実行するのではなく、特定のページが開かれたときにのみコードを実行するようにして、アプリケーションを 最適化できます。ユーザーがそのページを開かなければ、コードは実行されません。

以下の表はそれぞれのイベントごとにどのようにオブジェクトメソッドとフォームメソッドが呼ばれるかを概説します:

イベント	オブジェクトメソッド	フォームメソッド	オブジェクト
On Load	O	O	すべてのオブジェクト
On Unload	O	O	すべてのオブジェクト
On Validate	O	O	すべてのオブジェクト
On Clicked	O (クリック可能または入力可能なら) (*)	O	関係するオブジェクトのみ
On Double Clicked	O (クリック可能または入力可能なら) (*)	O	関係するオブジェクトのみ
On Before Keystroke	O (入力可能なら) (*)	O	関係するオブジェクトのみ
On After Keystroke	O (入力可能なら) (*)	O	関係するオブジェクトのみ
On After Edit	O (入力可能なら) (*)	O	関係するオブジェクトのみ
On Getting Focus	O (フォーカス可なら) (*)	O	関係するオブジェクトのみ
On Losing Focus	O (フォーカス可なら) (*)	O	関係するオブジェクトのみ
On Activate	X	O	None
On Deactivate	X	O	None
On Outside Call	X	O	None
On Page Change	X	O	None
On Begin Drag Over	O (ドラッグ可なら) (**)	O	関係するオブジェクトのみ
On Drop	O (ドロップ可なら) (**)	O	関係するオブジェクトのみ
On Drag Over	O (ドロップ可なら) (**)	Never	関係するオブジェクトのみ
On Mouse Enter	O	O	すべてのオブジェクト
On Mouse Move	O	O	すべてのオブジェクト
On Mouse Leave	O	O	すべてのオブジェクト
On Menu Selected	X	O	None
On Mac toolbar button	X	O	None
On bound variable change	X	O	None
On Data Change	O (更新可なら)) (*)	O	関係するオブジェクトのみ
On Plug in Area	O	O	関係するオブジェクトのみ
On Header	O	O	すべてのオブジェクト
On Printing Detail	O	O	すべてのオブジェクト
On Printing Break	O	O	すべてのオブジェクト
On Printing Footer	O	O	すべてのオブジェクト
On Close Box	X	O	None
On Display Detail	O	O	すべてのオブジェクト
On Open Detail	X (ただしリストボックス以外)	Yes	リストボックスのみ
On Close Detail	X (ただしリストボックス以外)	Yes	リストボックスのみ
On Open Detail	X	O	None
On Close Detail	X	O	None
On Resize	X	O	None
On Selection Change	O (***)	O	関係するオブジェクトのみ
On Load Record	X	O	None
On Timer	X	O	None
On Picture Scroll	O	O	関係するオブジェクトのみ
On Before Data Entry	O (リストボックス)	X	関係するオブジェクトのみ
On Column Moved	O (リストボックス)	X	関係するオブジェクトのみ
On Row Moved	O (リストボックス)	X	関係するオブジェクトのみ
On Column Resize	O (リストボックス)	X	関係するオブジェクトのみ
On Header Click	O (リストボックス)	X	関係するオブジェクトのみ
On Footer Click	O (リストボックス)	X	関係するオブジェクトのみ
On After Sort	O (リストボックス)	X	関係するオブジェクトのみ
On Long Click	O (3D ボタン)	O	関係するオブジェクトのみ
On Arrow Click	O (3D ボタン)	O	関係するオブジェクトのみ
On Expand	O (階層リストとリストボックス)	X	関係するオブジェクトのみ
On Collapse	O (階層リストとリストボックス)	X	関係するオブジェクトのみ
On Delete Action	O (階層リストとリストボックスx)	X	関係するオブジェクトのみ
On URL Resource Loading	O (Web エリア)	X	関係するオブジェクトのみ
On Begin URL Loading	O (Web エリア)	X	関係するオブジェクトのみ
On URL Loading Error	O (Web エリア)	X	関係するオブジェクトのみ
On URL Filtering	O (Web エリア)	X	関係するオブジェクトのみ
On End URL Loading	O (Web エリア)	X	関係するオブジェクトのみ
On Open External Link	O (Web エリア)	X	関係するオブジェクトのみ
On Window Opening Denied	O (Web エリア)	X	関係するオブジェクトのみ

(*) 詳細は後述の "イベント、オブジェクト、プロパティ" を参照
(**) 詳細は"ドラッグ＆ドロップ"を参照
(***) リストボックス、階層リスト、サブフォーム型のオブジェクトのみがこのイベントをサポートします

重要: イベントに対応するプロパティが選択されている場合にのみ、フォームやオブジェクトのメソッドが呼び出されることに留意してください。デザインモードの フォームエディタのプロパティリストでイベントを無効にすると、メソッドが呼び出される回数を減らすことができ、フォームの実行速度を最適化できます。

警告: オブジェクトのOn Load と On Unload イベントが生成されるには、オブジェクトとオブジェクトが属するフォームの両方で有効にされていなければなりません。オブジェクトのみでイベントが有効に なっている場合、イベントは生成されません。これら2つのイベントはフォームレベルでも有効にされていなければなりません。

イベント、オブジェクト、プロパティ

イベントの性質やプロパティに基づき、オブジェクトに対してイベントが生成されるとオブジェクトメソッドが呼び出されます。ここでは、さまざまなタイプのオブジェクトを扱うにあたり、一般的に使用されるイベントについて詳細に説明します。

フォームエディタのプロパティリストには、選択されたオブジェクトあるいはフォームで利用可能なイベントのみが表示されることに留意してください。

クリック可能なオブジェクト

クリック可能なオブジェクトは主にマウスで操作します。以下のオブジェクトが含まれます:

- ブールの入力可能フィールド/変数
- ボタン, デフォルトボタン, ラジオボタン, チェックボックス, ボタングリッド
- 3D ボタン, 3D ラジオボタン, 3D チェックボックス
- ポップアップpメニュー, 階層ポップアップメニュー, ピクチャメニュー
- ドロップダウンリスト, メニュー/ドロップダウンリスト
- スクロールエリア, 階層リスト, リストボックス
- 非表示ボタン, ハイライトボタン, ラジオピクチャ
- サーモメータ, ルーラ, ダイアル (スライダオブジェクト)
- タブコントロール
- スプリッタ

On Clicked や On Double Clicked オブジェクトイベントプロパティを選択したのち、Form event コマンドを使用してオブジェクト上でのクリックを検知し処理することができます。Form event コマンドはユーザアクションに応じ、On Clicked または On Double Clickedを返します。

注: 4D v14以降、テキスト(テキスト、日付、時間、数字型)が含まれる入力可能なフィールドおよび変数は On Clicked と On Double Clicked イベントを生成するようになりました。

両イベントがオブジェクトに対し選択されている場合、ダブルクリックが行われるとまずOn Clicked が、そしてOn Double Clicked イベントが生成されます。

これらすべてのオブジェクトにおいて、On Clickedイベントはマウスボタンが離されたときに生成されます。しかしいくつか例外があります:
・非表示ボタン - マウスがクリックされると、ボタンが離されるのを待たずにOn Clickedイベントが生成されます。
・スライダオブジェクト (サーモメータ, ルーラ, ダイアル) - 表示フォーマットでコントロールをスライド中にオブジェクトメソッドが呼び出されるように設定されていると、On Clickedイベントはクリックが行われるとすぐに生成されます。

注: いくつかのオブジェクトはキーボードからも操作可能です。例えばチェックボックスがフォーカスを得ると、スペースバーでオン/オフを切り替えることができます。この場合でもOn Clickedイベントは生成されます。

警告: コンボボックスはクリック可能なオブジェクトとしては扱われません。コンボボックスは入力可能なテキストエリアとして扱われ、割り当てられたドロップダウンリストにはデフォルト値が提供されます。コンボボックスの処理はOn Before Keystroke, On After Keystroke そして On Data Change イベントを使用して行います。

注: 4D v13 以降、ポップアップメニュー/ドロップダウンリストと、階層ポップアップメニューもOn Data Change を生成できるようになりました。これにより、カレントの値と異なる値が選択されたときにオブジェクトの起動を察知できるようになりました。

キーボードから入力可能なオブジェクト

キーボードから入力可能なオブジェクトとは、キーボードを使用してデータを入力できるオブジェクトです。これらのオブジェクトではOn After Edit, On Before KeystrokeそしてOn After Keystrokeイベントを検知し、Get edited textコマンドを使用して、低レベルでデータ入力をフィルタできます。

キーボード入力可能なオブジェクトやデータタイプには以下が含まれます:

- 文字、テキスト、日付、時間、数値、そしてピクチャ(On After Edit のみ) など、すべての入力可能なフィールドオブジェクト
- 文字、テキスト、日付、時間、数値、そしてピクチャ(On After Edit のみ) など、すべての入力可能な変数
- コンボボックス
- リストボックス

注: 4D v14以降、テキスト(テキスト、日付、時間、数字型)が含まれる入力可能なフィールドおよび変数は On Clicked と On Double Clicked イベントを生成するようになりました。

注: 階層リストは入力可能なオブジェクトですが、このオブジェクトではOn After Edit, On Before KeystrokeそしてOn After Keystrokeは管理されません (後述の "階層リスト" の段落を参照)。

- On Before Keystroke と On After Keystroke

  注: On After Keystroke イベントは、一般的に On After Edit イベントで置き換えることができます(詳細後述)。

  On Before Keystroke と On After Keystroke イベントプロパティを選択したら、**Form event**コマンドを使用して返されるOn Before KeystrokeとOn After Keystrokeイベントを検知し、オブジェクトへのキーストロークを処理できます (詳細はGet edited textコマンドの説明を参照してください。これらのイベントはPOST KEYのようなユーザアクションをシミュレートするコマンドによっても生成されます。

  キーボードを使用せず、ペーストやドラッグ＆ドロップなどで行われた変更は考慮されないことに留意してください。これらのイベントを処理するためにはOn After Editを使用します。

  Note: On Before KeystrokeとOn After Keystroke イベントは入力メソッド (IME) 使用時は生成されません。入力メソッドはプログラムあるいはシステムコンポーネントで、日本語や中国語など特定の文字や記号を入力するために使用されます。
- On After Edit

  このイベントは、変更が行われた方法には関係なく、入力可能オブジェクトの内容に変更が行われるたびに生成されます。例えば:

  - ペーストやカット、削除、キャンセルなどの標準の編集アクション
  - 値のドロップ (ペーストと同様のアクション)
  - ユーザが行ったキーボードからの入力。この場合On After EditイベントはOn Before KeystrokeとOn After Keystrokeイベントの後に生成されます。
  - ユーザアクションをシミュレートするランゲージコマンドを使用した変更 (例 POST KEY)。

  以下のアクションはこのイベントを生成させないことに注意してください:
  - コピーやすべてのを選択のような、内容を変更しない編集アクション;
  - 値のドラッグ (コピーに該当するアクション);
  - プログラムにより行われた内容の変更、ただしユーザアクションをシミュレートするコマンドを除く。

  このイベントはユーザアクションをコントロールするために使用できます。例えば長すぎるテキストのペーストや特定の文字のブロック、パスワードフィールドに対するカットを防ぐなどです。
- On Selection Change:
  (入力可であるかそうでないかに関わらず) 動的なテキストフィールドや変数に適用されると、このイベントはカーソルの位置が変わるたびに生成されます。例えばユーザーがマウスやキーボードの矢印キーをを使用してテキストを選択したときや、ユーザーがテキストを入力したときです。ここでGET HIGHLIGHTのようなコマンドを呼び出すことができます。

変更可能オブジェクト

更新可能オブジェクトは、マウスやキーボードを使用して値を変更することのできるデータソースを持ちます。これらはOn Clickedイベントで処理されるユーザインタフェースコントロールとしては扱われません。以下が含まれます:

- すべての入力可フィールドオブジェクト (Blobを除く)
- すべての入力可変数 (Blob、ポインタ、配列を除く)
- コンボボックス
- 外部オブジェクト (完全なデータ入力が許可されるプラグイン)
- 階層リスト
- リストボックス

これらのオブジェクトはOn Data Changeイベントを受け取ります。On Data Changeオブジェクトイベントプロパティが選択されると、Form event コマンドから返されるOn Data Changeによりイベントを検知し、データソース値の変更を処理できます。イベントは、オブジェクトに結び付けられた変数が4Dにより内部的に更新され次第、生成されます (例えば、一般的に入力エリアオブジェクトがフォーカスを失った時)。

タブ可オブジェクト

タブ 可オブジェクトは、タブキーやクリックによりフォーカスを得るオブジェクトです。フォーカスを持つオブジェクトは、メニューやボタンへのモディファイアではない (キーボードでタイプされた) 文字を受け取ります。

以下を除き、すべてのオブジェクトはタブ可です:

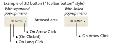
- 入力不可フィールドや変数
- ボタングリッド
- 3D ボタン, 3D ラジオボタン, 3D チェックボックス
- ポップアップメニュー, 階層ポップアップメニュー
- メニュー/ドロップダウンリスト
- ピクチャメニュー
- スクロールエリア
- 非表示ボタン, ハイライトボタン, ラジオピクチャボタン
- グラフ
- 外部オブジェクト (完全なデータ入力が許可されるプラグイン)
- タブコントロール
- スプリッタ.

On Getting Focus や On losing Focus オブジェクトイベントプロパティを選択したら、Form event コマンドから返されるOn Getting Focus や On Losing Focusを検知して、フォーカスの変更を処理できます。

3D ボタン

3D を使用すると詳細にグラフィックインタフェースを設定できます (f3D ボタンに関する説明はDesign Referenceマニュアルを参照してください)。汎用のイベントに加え、これらのボタンを管理するために2つの追加のイベントを使用できます:

- On Long Click: このイベントは3D ボタンがクリックされ、一定時間以上マウスボタンが押され続けていると生成されます。理論上、このイベントが生成されるためのクリック保持時間は、システムの環境設定に設定されたダブルクリックの間隔最大時間に等しくなります。
  このイベントはすべての3D ボタンスタイル、3D ラジオボタン、3D チェックボックスで生成されます。例外は、旧世代の3D ボタンであるバックグランドオフセットタイプと、ポップアップメニューが関連付けられた3D ボタンの矢印エリアです (後述)。
  このイベントは一般的にロングクリック時にポップアップメニューを表示するために使用します。ユーザがロングクリックが有効になる時間前にマウスボタンを離すと、On Clickedが生成されます。
- On Arrow Click: いくつかの3D ボタンスタイルには、ポップアップメニューをリンクし、矢印を表示させることができます。この矢印をクリックすると、ボタンの主たるアクションに追加してアクションを提供するポップアップを表示します。
  4DではOn Arrow Clickイベントを使用してこの動作を管理できます。このイベントはユーザが矢印をクリックすると、マウスボタンが押されてすぐに生成されます:
  - ポップアップメニューが分離されている場合、このイベントはボタン中で矢印のあるエリアがクリックされた場合のみ生成されます。
  - ポップアップメニューがリンクされている場合、このイベントはボタン上どこをクリックしても生成されます。このタイプのボタンではOn Long Clickイベントが生成されないことに注意してください。



なし, ツールバーボタン, ベベル, 角の丸いベベルおよびOffice XPタイプの3D ボタンや、3D ラジオボタン、3D チェックボックスは"ポップアップメニューあり"プロパティをサポートします:

リストボックス

リストボックス特有の様々な機能を管理するために特別なフォームイベントを使用できます:

- On Before Data Entry: このイベントは、リストボックス中のセルが編集される直前に生成されます (入力カーソルが表示される前)。このイベントを使用して、例えば表示中と編集中で異なるテキストを表示させることができます。
- On Selection Change: このイベントは、リストボックスの列や行の現在の選択が変更されるたびに生成されます。このイベントはレコードリストや階層リストでも生成されます。
- On Column Moved: このイベントは、ユーザのドラッグ＆ドロップでリストボックスの列が移動されたときに生成されます。ただし元の場所にドロップされた場合には生成されません。LISTBOX MOVED COLUMN NUMBERコマンドは列の新しい位置を返します。
- On Row Moved: このイベントは、ユーザのドラッグ＆ドロップでリストボックスの行が移動されたときに生成されます。ただし元の場所にドロップされた場合には生成されません。
- On Column Resize: このイベントは、(マウスやLISTBOX SET COLUMN WIDTHコマンドを使用して) リストボックス中の列幅が変更されたときに生成されます。
- On ExpandとOn Collapse: これらのイベントは階層リストボックスの行が展開されたり折りたたまれたときに生成されます。
- On Header Click: このイベントは、リストボックス中の列ヘッダでクリックが行われると生成されます。この場合Self コマンドを使用すればクリックされた列ヘッダを知ることができます。右クリック (Windouws) やCtrl+クリック (Mac OS) を列や列ヘッダ上で行うと、On Clickedイベントが生成されます。
  リストボックスで**並び替え可**プロパティが選択されている場合、$0に0または-1を渡して標準の並び替えを行うかどうか指定できます:
  - $0 = 0の場合、標準の並び替えが行われます。
  - $0 = -1の場合、標準の並び替えは行われず、ヘッダには並び替え矢印は表示されません。開発者は4Dの配列コマンドを使用して、カスタマイズされた条件に基づく並び替えを実行できます。
  並び替え可 プロパティが選択されていない場合、$0は使用されません。
- On Footer Click: このイベントはリストボックスやリストボックス列で利用でき、リストボックスやリストボックス列のフッターエリアがクリックされたときに生成されます。この場合 OBJECT Get pointer コマンドはクリックされたフッター変数へのポインターを返します。イベントは左および右クリックどちらでも生成されます。
- On After Sort: このイベントは標準の並び替えが行われた直後に生成されます (On Header Clickイベントで$0に-1が返された場合には生成されません)。このメカニズムは、ユーザによって行われた最後の並び替えの方向を格納するために使用できます。このイベント内でSelfコマンドは、並び替えられた列変数へのポインタを返します。
- On Delete Action: このイベントはユーザーが削除キー (DeleteやBackspaceキー) を押して、またはクリア標準アクションが割り当てられたメニュー項目 (編集メニューのクリア等) を選択して、選択された行の削除を指示したときに生成されます。このイベントはリストボックスオブジェクトレベルでのみ利用できます。4Dはイベントの生 成を行うだけであることに留意してください。4Dは項目を消去しません。実際の削除処理や事前警告の表示などは開発者の責任です。

(v14からの新機能) 「セレクション」型のリストボックスのコンテキストにおいて、二つの一般的なイベントが使用できるようになりました:

- On Open Detail: このイベントはレコードが「セレクション」型リストボックスに関連付けられた(まだ開かれていない)詳細フォームに表示されるときに生成されます。
- On Close Detail: このイベントは「セレクション」型リストボックスに関連付けられた詳細フォームに表示されたレコードが閉じられるときに生成されます(レコードが変更されたかどうかは関係しません)。

階層リスト

汎用のイベントに加え、階層リスト上で行われるアクションを処理するために3つのイベントを使用できます:

- On Selection Change: このイベントは、マウスクリックやキーストロークで階層リスト中の選択が変更されるたびに生成されます。
  このイベントはリストボックスやレコードリストでも生成されます。
- On Expand: このイベントは、マウスクリックやキーストロークで階層リストの要素が拡げられるたびに呼び出されます。
- On Collapse: このイベントは、マウスクリックやキーストロークで階層リストの要素が折りたたまれるたびに呼び出されます。
- On Delete Action: このイベントはユーザーが削除キー (DeleteやBackspaceキー) を押して、またはユーザーがクリア標準アクションが割り当てられたメニュー項目 (編集メニューのクリア等) を選択して、選択された行の削除を指示したときに生成されます。このイベントはリストボックスオブジェクトレベルでのみ利用できます。4Dはイベントの生成を行うだけであることに留意してください。4Dは項目を消去しません。実際の削除処理や事前警告の表示などは開発者の責任です。

これらのイベントは相互に排他的ではなく、階層リスト中で連続して生成されることがあります:
- キーストロークに伴い (順に):

イベント	**コンテキスト**
On Data Change	要素が編集された
On Expand/On Collapse	-> または <-矢印キーを使用してサブリストを開いた/閉じた
On Selection Change	新しい要素を選択した
On Clicked	キーボードを使用してリストをアクティブにした

- マウスクリックに伴い (順に):

イベント	**コンテキスト**
On Data Change	要素が編集された
On Expand/On Collapse	拡げる/折りたたむアイコンを使用してを使用してサブリストを開いた/閉じた または 編集不可サブリストをダブルクリックした
On Selection Change	新しい要素を選択した
On Clicked / On Double Clicked	クリックまたはダブルクリックでリストをアクティブにした

ピクチャーフィールドと変数

(4D v13.2より) On Picture Scroll フォームイベントはユーザーが (フィールドや変数) エリア内のピクチャーをスクロールすると生成されます。ピクチャーエリアのサイズがピクチャーよりも小さく、かつ表示フォーマットが"**トランケート (中央合わせなし)**"に設定されている場合のみスクロールが可能です。この点についてはピクチャーフォーマットを参照してください。

このイベントはスクロールがユーザーアクション (スクロールバーやカーソル、マウスホイール、キーボードの利用) の結果として行われた場合に生成されます (キーボードによるスクロールについてはスクロールバー参照)。

注: OBJECT SET SCROLL POSITIONコマンドの結果としてスクロールが行われた場合、イベントは生成されません。

例題15を参照してください。

サブフォーム

サブフォームコンテナオブジェクト (親フォーム上に置かれた、ひとつのサブフォームインスタンスを含むオブジェクト) は以下のイベントをサポートします:

- On Load と On Unload: サーブフォームを開くまた閉じる際にそれぞれ生成されます。これらのイベントは親フォームレベルでも有効にされていなければなりません。これらのイベント は親のフォームの同じイベントよりも前に生成される点に留意してください。またフォーイベント動作の原則にいたがい、サブフォームが0もしくは1ページ以 外のページに配置されている場合、これらのイベントはページが閉じられ開かれるときにに生成され、フォームが開かれ閉じられるときではないことに留意して ください。
- On Validate: サブフォーム中でデータの受け入れを行う際。
- On Data Change: サブフォームオブジェクト変数の値が更新されたとき。
- On Getting Focus and On Losing Focus: サブフォームコンテナがフォーカスを得たとき、また失った時。これらのイベントはプロパティリスト中でチェックされていれば、サブフォームオブジェクトの メソッド内で生成されます。これらはサブフォームのフォームメソッドに送信されます。つまり例えば、フォーカスに応じてサブフォーム中のナビゲーションボ タンの表示を管理できます。
  サブフォームオブジェクトはそれ自身がフォーカスを持つ点に留意してください。
- On Bound Variable Change: この特別なイベントは、親フォーム中のサブフォームにバインドされた変数に割り当てられた値が更新される (同じ値が割り当てられたばあいでも) 、かつサブフォームがカレントフォームページまたは0ページに属していれば、サブフォームメソッドのコンテキストで生成されます。サブフォームの管理につ いては*Design Reference*マニュアルを参照してください。

Note: CALL SUBFORM CONTAINER コマンドを使用して、サブフォーム内で生成可能なあらゆるカスタムイベントタイプを指定できます。このコマンドにより、コンテナオブジェクトメソッドを呼び出し、イベントコードを渡すことができます。

注: サブフォームにて生成された On Clicked と On Double Clicked イベントは、最初にフォームメソッドで受け取られ、その後サブフォームのメソッドに、そしてホストフォームのフォームメソッドへと受け取られます。

Webエリア

Webエリアでは7つのイベントを利用できます:

- On Begin URL Loading: このイベントは、Webエリアに新しいURLのロードを開始した時に生成されます。Webエリアに関連付けられた"URL"変数を使用してロード中のURLを知ることができます。
  Note: ロード中のURLは現在のURLとは異なります (WA Get current URL コマンドの説明参照)。
- On URL Resource Loading: このイベントは、現在のWebページに (ピクチャやフレームなど) 新しいリソースをロードするたびに生成されます。Webエリアに関連付けられた"Progression"変数を使用してロード状況を知ることができます。
- On End URL Loading: このイベントは、現在のURLのすべてのリソースがロードされると生成されます。
  WA Get current URL コマンドを使用して、ロードされたURLを知ることができます。
- On URL Loading Error: このイベントは、URLのロード中にエラーを検出すると生成されます。
  WA GET LAST URL ERROR コマンドを使用して、エラーに関する情報を取得できます。
- On URL Filtering: このイベントは、WA SET URL FILTERS コマンドで設定されたフィルタにより、URLのロードがWebエリアによってブロックされると生成されます。
  WA Get last filtered URL コマンドコマンドを使用してブロックされたURLを知ることができます。
- On Open External Link: このイベントは、WA SET EXTERNAL LINKS FILTERS コマンドで設定されたフィルタにより、URLのロードがWebエリアによってブロックされ、URLがカレントのシステムブラウザで開かれると生成されます。
  WA Get last filtered URL コマンドコマンドを使用してブロックされたURLを知ることができます。
- On Window Opening Denied: このイベントは、Webエリアによりポップアップウィンドウがブロックされると生成されます。4D Web エリアはポップアップウィンドウを許可しません。
  WA Get last filtered URL コマンドコマンドを使用してブロックされたURLを知ることができます。

例題 1

この例題ではレコード更新日をOn Validateイベントで自動的に割り当てる例を示します:

```
  ` Method of a form
 Case of
  ` ...
    :(Form event=On Validate)
       [aTable]Last Modified On:=Current date
 End case
```

例題 2

この例題では、ドロップダウンリスト処理 (初期化, ユーザクリック, オブジェクトのリリース) をオブジェクトメソッドにカプセル化します:

```
  ` asBurgerSize Drop-down list Object Method
 Case of
    :(Form event=On Load)
       ARRAY STRING(31;asBurgerSize;3)
       asBurgerSize{1}:="Small"
       asBurgerSize{1}:="Medium"
       asBurgerSize{1}:="Large"
    :(Form event=On Clicked)
       If(asBurgerSize#0)
          ALERT(asBurgerSize{asBurgerSize}+" バーガーが選択されました。")
       End if
    :(Form event=On Unload)
       CLEAR VARIABLE(asBurgerSize)
 End case
```

例題 3

この例題ではオブジェクトメソッドで、ピクチャのみを受け付けるフィールドのドラッグ＆ドロップ処理の方法を示します:

```
  ` [aTable]aPicture ピクチャ フィールドオブジェクトメソッド
 Case of
    :(Form event=On Drag Over)
  ` ドラッグ＆ドロップ処理が開始され、マウスが現在フィールド上にある
  ` ソースオブジェクトに関する情報を取得
       DRAG AND DROP PROPERTIES($vpSrcObject;$vlSrcElement;$lSrcProcess)
  ` ソースプロセスの番号をテストする必要はない
  ` 実行されているオブジェクトメソッドは同じプロセス内で動作している
       $vlDataType:=Type($vpSrcObject->)
  ` ソースデータはピクチャか (フィールド, 変数または配列)?
       If(($vlDataType=Is Picture)|($vlDataType=Picture array))
  ` 真ならドラッグを受け入れる
          $0:=0
       Else
  ` 偽ならドラッグを拒否する
          $0:=-1
       End if
    :(Form event=On Drop)
  ` ソースデータがオブジェクトにドロップされたのでオブジェクトにコピーする
  ` ソースオブジェクトに関する情報を取得
       DRAG AND DROP PROPERTIES($vpSrcObject;$vlSrcElement;$lSrcProcess)
       $vlDataType:=Type($vpSrcObject->)
       Case of
  ` ソースオブジェクトはピクチャフィールドまたは変数
          :($vlDataType=Is Picture)
  ` ソースオブジェクトは同じプロセスから来ているか (つまり同じウィンドウのフォームか)t?
             If($lSrcProcess=Current process)
  ` そうならソースをコピー
                [aTable]aPicture:=$vpSrcObject->
             Else
  ` そうでない場合、ソースオブジェクトを利用できるか?
                If(Is a variable($vpSrcObject))
  ` 真ならソースプロセスから値を取得
                   GET PROCESS VARIABLE($lSrcProcess;$vpSrcObject->;$vgDraggedPict)
                   [aTable]aPicture:=$vgDraggedPict
                Else
  ` 偽ならCALL PROCESSを使用してソースプロセスから値を取得
                End if
             End if
  ` ソースオブジェクトがピクチャ配列
          :($vlDataType=Picture array)
  ` ソースオブジェクトは同じプロセスにあるか (つまり同じプロセスの同じウィンドウか)?
             If($lSrcProcess=Current process)
  ` 真ならソース値をコピー
                [aTable]aPicture:=$vpSrcObject->{$vlSrcElement}
             Else
  ` そうでなければソースプロセスから値を取得
                GET PROCESS VARIABLE($lSrcProcess;$vpSrcObject
                ->{$vlSrcElement};$vgDraggedPict)
                [aTable]aPicture:=$vgDraggedPict
             End if
       End case
 End case
```

Note: On Drag Over や On Drop イベントを使用する他の例題は、DRAG AND DROP PROPERTIES コマンドの例題を参照してください。

例題 4

この例題はフォームメソッドのテンプレートです。出力フォームとしてサマリレポートがフォームを使用する際に発生するイベントを示しています:

```
  ` Method of a form being used as output form for a summary report
 $vpFormTable:=Current form table
 Case of
  ` ...
    :(Form event=On Header)
  ` ヘッダエリアの印刷開始
       Case of
          :(Before selection($vpFormTable->))
  ` 最初のブレークヘッダ用のコード
          :(Level=1)
  ` ヘッダブレークレベル 1 用のコード
          :(Level=2)
  ` ヘッダブレークレベル 2 用のコード
  ` ...
       End case
    :(Form event=On Printing Detail)
  ` レコードの印刷開始
  ` レコード毎のコードを記述
    :(Form event=On Printing Break)
  ` ブレークエリアの印刷開始
       Case of
          :(Level=0)
  ` ブレークレベル0 用のコード
          :(Level=1)
  ` レークレベル1 用のコード
  ` ...
       End case
    :(Form event=On Printing Footer)
       If(End selection($vpFormTable->))
  ` 最後のフッタ用のコード
       Else
  ` フッタ用のコード
       End if
 End case
```

例題 5

この例題はDISPLAY SELECTIONやMODIFY SELECTIONで表示されるフォームで発生するイベントを処理するメソッドのテンプレートです。説明的にするため、フォームウィンドウのタイトルバーにイベントの説明が表示されます:

```
  ` フォームメソッド
 Case of
    :(Form event=On Load)
       $vsTheEvent:="フォームが表示されようとしている"
    :(Form event=On Unload)
       $vsTheEvent:="出力フォームを抜け、スクリーンから消えようとしている"
    :(Form event=On Display Detail)
       $vsTheEvent:="表示中のレコード #"+String(Selected record number([TheTable]))
    :(Form event=On Menu Selected)
       $vsTheEvent:="メニュー項目が選択された"
    :(Form event=On Header")
       $vsTheEvent:="ヘッダエリアが描画されようとしている"
    :(Form event=On Clicked")
       $vsTheEvent:="レコードがクリックされた"
    :(Form event=On Double Clicked")
       $vsTheEvent:="レコードがダブルクリックされた"
    :(Form event=On Open Detail)
       $vsTheEvent:="レコード #"+String(Selected record number([TheTable]))+" がダブルクリックされた"
    :(Form event=On Close Detail)
       $vsTheEvent:="出力フォームに戻る"
    :(Form event=On Activate)
       $vsTheEvent:="フォームのウィンドウが最前面になった"
    :(Form event=On Deactivate)
       $vsTheEvent:="フォームのウィンドウが最前面でなくなった"
    :(Form event=On Outside Call)
       $vsTheEvent:="プロセスの呼び出しを受信した"
    Else
       $vsTheEvent:="発生したイベント #"+String(Form event)
 End case
 SET WINDOW TITLE($vsTheEvent)
```

例題 6

On Before Keystroke と On After Keystroke イベントを処理する方法はGet edited text、Keystroke、そしてFILTER KEYSTROKEコマンドの説明を参照してください。

例題 7

この例題は、スクロールエリアでクリックとダブルクリックを同様に扱う方法を示しています:

```
  ` asChoices scrollable area object method
 Case of
    :(Form event=On Load)
       ARRAY STRING(...;asChoices;...)
  ` ...
       asChoices:=0
    :((Form event=On Clicked)|(Form event=On Double Clicked))
       If(asChoices#0)
  ` 項目がクリックされたので、何らかの処理を行う
  ` ...
       End if
  ` ...
 End case
```

例題 8

この例題では、クリックとダブルクリックで異なるレスポンスをする方法を示します。要素0を使用して選択された項目を追跡しています:

```
  ` asChoices scrollable area object method
 Case of
    :(Form event=On Load)
       ARRAY STRING(...;asChoices;...)
  ` ...
       asChoices:=0
```

```
      asChoices{0}:="0"
   :(Form event=On Clicked)
      If(asChoices#0)
         If(asChoices#Num(asChoices))
  ` 新しい項目がクリックされたので何かを行う
  ` ...
  ` 次回のために、新しく選択された要素を保存
            asChoices{0}:=String(asChoices)
         End if
      Else
         asChoices:=Num(asChoices{0})
      End if
   :(Form event=On Double Clicked)
      If(asChoices#0)
  ` 項目がダブルクリックされたのでここで何かを行う
      End if
  ` ...
 End case
```

例題 9

この例題では、On Getting Focus と On Losing Focusを使用して、フォームメソッド内でステータス情報を管理します。:

```
  ` [Contacts];"Data Entry" form method
 Case of
   :(Form event=On Load)
      C_TEXT(vtStatusArea)
      vtStatusArea:=""
   :(Form event=On Getting Focus)
      RESOLVE POINTER(Focus object;$vsVarName;$vlTableNum;$vlFieldNum)
      If(($vlTableNum#0) & ($vlFieldNum#0))
         Case of
            :($vlFieldNum=1) ` Last name フィールド
               vtStatusArea:="Enter the Last name of the Contact; it will be capitalized automatically"
  ` ...
            :($vlFieldNum=10) ` Zip Code フィールド
               vtStatusArea:="Enter a 5-digit zip code; it will be checked and validated automatically"
  ` ...
         End case
      End if
   :(Form event=On Losing Focus)
      vtStatusArea:=""
  ` ...
 End case
```

例題 10

この例題では、レコードデータ入力に使用されるフォームで、ウィンドウを閉じる際の処理を示します:

```
  ` Method for an input form
 $vpFormTable:=Current form table
 Case of
  ` ...
   :(Form event=On Close Box)
      If(Modified record($vpFormTable->))
         CONFIRM("このレコードは更新されています。保存しますか?")
         If(OK=1)
            ACCEPT
         Else
            CANCEL
         End if
      Else
         CANCEL
      End if
  ` ...
 End case
```

例題 11

この例題では、文字フィールドが更新されるたびに、1文字目を大文字に、それ以外を小文字に変換する方法を示します:

```
  ` [Contacts]First Name Object method
 Case of
  ` ...
   :(Form event=On Data Change)
      [Contacts]First Name:=Uppercase(Substring([Contacts]First Name;1;1))+
      Lowercase(Substring([Contacts]First Name;2))
  ` ...
 End case
```

例題 12

この例題では、文字フィールドが更新されるたびに、1文字目を大文字に、それ以外を小文字に変換する方法を示します:

```
  ` [Contacts]First Name Object method
 Case of
  ` ...
   :(Form event=On Data Change)
      [Contacts]First Name:=Uppercase(Substring([Contacts]First Name;1;1))+
      Lowercase(Substring([Contacts]First Name;2))
  ` ...
 End case
```

例題 13

以下の例題では階層リストで削除アクションを管理する方法を示します:

```
 ... // 階層リストメソッド
:($Event=On Delete Action)
 ARRAY LONGINT($itemsArray;0)
 $Ref:=Selected list items(<>HL;$itemsArray;*)
 $n:=Size of array($itemsArray)

 Case of
   :($n=0)
      ALERT("項目が選択されていません。")
      OK:=0
   :($n>=1)
      CONFIRM("選択された項目を削除しますか?")
 End case

 If(OK=1)
   For($i;1;$n)
      DELETE FROM LIST(<>HL;$itemsArray{$i};*)
   End for
 End if
```

例題 14

この例題では On Picture Scroll フォームイベントを使用してフォーム中の2つのピクチャーを同期します。以下のコードをフォームメソッドに記述します:

```
 Case of
   :(Form event=On Picture Scroll)
  // 左側のピクチャー位置を取得
      OBJECT GET SCROLL POSITION(*;"satellite";vPos;hPos)
  // 右側のピクチャーに適用
      OBJECT SET SCROLL POSITION(*;"plan";vPos;hPos;*)
 End case
```

結果:

Right click

Right click -> 戻り値

引数	型		説明
戻り値	ブール	←	右クリックを検知した場合True、そうでなければFalse

説明

Right click コマンドは、マウスの右ボタンがクリックされた場合にTrue を返します。

このコマンドはOn clickedフォームイベントのコンテキストで使用します。したがって、デザインモードにおいて、このイベントがフォームや特定のオブジェクトのプロパティで適切に選択されていることを確認する必要があります。

SET TIMER

SET TIMER (tickCount)

引数	型	説明
tickCount	倍長整数	→ Tickcount または -1=すぐに実行する

説明

SET TIMERコマンドは、On Timerフォームイベントを有効にし、カレントプロセスのカレントフォームでOn Timerフォームイベント間の間隔Tick数を設定します。

Note: このフォームイベントに関する詳細は、Form eventコマンドの説明を参照してください。

このコマンドをフォームを表示していないコンテキストで呼び出しても、効果はありません。

Note: SET TIMER コマンドがサブフォームのコンテキスト (サブフォームメソッド) で呼び出されると、On Timer イベントは親フォームレベルではなくサブフォームで生成されます。

tickCount 引数に-1を渡すと、コマンドはOn Timer フォームイベントを即座に有効にします。言い換えれば、4Dアプリケーションがイベントマネージャにコントロールを渡し次第ということです。特にこれは、処理の開始前にフォームが完全に表示されることを意味します。

On Timerフォームイベントの生成を取り消すには、tickCountに0をセットしたSET TIMERをもう一度実行してください。

例題

フォームが画面に表示されている時に、コンピュータが3秒毎に警告音を鳴らすようにしたいと仮定します。これを実行するには、下記のようにフォームメソッドを書きます:

```
If(Form event=On Load)
   SET TIMER(60*3)
End if

If(Form event=On Timer)
   BEEP
End if
```

Activated

Activated -> 戻り値

引数	型		説明
戻り値	ブール	←	実行サイクルがactivationである場合にTrueを返す

互換性に関するメモ

このコマンドは互換性を維持する目的で残されています。バージョン6からは、Form eventを使用し、On Activateイベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

説明

Activatedコマンドは、フォームを含むウインドウがプロセスの最前面のウインドウになると、そのフォームメソッドでTrueを返します。

警告: フォームのActivatedフェーズにTRACEまたはALERTを置かないでください。入れると無限ループになります。

Note: Activated 実行サイクルを生成させるには、デザインモードでそのフォームのOn Activateイベントプロパティを必ず選択してください。データベースを変換した際に、この作業は自動的に実行されます。

After

After -> 戻り値

引数	型		説明
戻り値	ブール	↩	Returns True if the execution cycle is an after

互換性に関するメモ

このコマンドは互換性を維持する目的で残されています。バージョン6からは、Form eventを使用し、On Validateイベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

説明

After実行サイクルを生成させるには、デザインモードでそのフォームやオブジェクトのOn Validateイベントプロパティを必ず選択してください。

Before

Before -> 戻り値

引数	型		説明
戻り値	ブール	←	Returns True if the execution cycle is a before

互換性に関するメモ

このコマンドは互換性を維持する目的で残されています。バージョン6からは、Form eventを使用し、On Loadイベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

説明

Before 実行サイクルを生成させるには、デザインモードでそのフォームやオブジェクトのOn Loadイベントプロパティを必ず選択してください。

Deactivated

Deactivated -> 戻り値

引数	型	説明
戻り値	ブール	↩ 実行サイクルがdeactivationである場合にTrueを返す

説明

このコマンドは互換性を維持する目的で残されています。バージョン6からは、Form eventを使用し、On Deactivateイベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

説明

Deactivatedコマンドはプロセスの最前面のウインドウが後ろに移動すると、そのフォームメソッドでTRUE を返します。

Deactivated実行サイクルを生成させるには、デザインモードでそのフォームやオブジェクトのOn Deactivateイベントプロパティを必ず選択してください。

During

During -> 戻り値

引数	型		説明
戻り値	ブール	↩	Returns True if the execution cycle is during

説明

このコマンドは互換性を維持する目的で残されています。バージョン6からは、Form eventを使用し、On Clickedイベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

説明

During 実行サイクルを生成させるには、デザインモードでそのフォームやオブジェクトの適切なフォームイベントプロパティ (On Clicked等) を必ず選択してください。

In break

In break -> 戻り値

引数	型		説明
戻り値	ブール	↩	Returns True if the execution cycle is in break

互換性に関するメモ

このコマンドは互換性を維持する目的で残されています。バージョン6からは、Form eventを使用し、On Printing Breakイベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

説明

In break実行サイクルを生成させるには、デザインモードでそのフォームやオブジェクトでOn Printing Breakイベントプロパティを必ず選択してください。

In footer

In footer -> 戻り値

引数	型	説明
戻り値	ブール	↩ Returns True if the execution cycle is in footer

互換性に関するメモ

このコマンドは互換性を維持する目的で残されています。バージョン6からは、Form eventを使用し、On Printing Footerイベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

説明

In footer 実行サイクルを生成させるには、デザインモードでそのフォームやオブジェクトでOn Printing footerイベントプロパティを必ず選択してください。

In header

In header -> 戻り値

引数	型		説明
戻り値	ブール	⮐	Returns True if the execution cycle is in header

互換性に関するメモ

このコマンドは互換性を維持する目的で残されています。バージョン6からは、Form eventを使用し、On Headerイベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

説明

In header 実行サイクルを生成させるには、デザインモードでそのフォームやオブジェクトで On Headerイベントプロパティを必ず選択してください。

Outside call

Outside call -> 戻り値

引数	型		説明
戻り値	ブール	←	True if the execution cycle is an outside call

互換性に関するメモ

このコマンドは互換性を維持する目的で残されています。バージョン6からは、Form eventを使用し、On Outside callイベントが返されるかどうかを調べる方法に切り替えることをお勧めします。

説明

Outside call 実行サイクルを生成させるには、デザインモードでそのフォームやオブジェクトで On Outside callイベントプロパティを必ず選択してください。

プロセス

- プロセス
- Count tasks
- Count user processes
- Count users
- Current process
- DELAY PROCESS
- EXECUTE ON CLIENT
- Execute on server
- GET REGISTERED CLIENTS
- New process
- PAUSE PROCESS
- Process aborted
- Process number
- PROCESS PROPERTIES
- Process state
- REGISTER CLIENT
- RESUME PROCESS
- UNREGISTER CLIENT

プロセス

4Dのマルチタスク機能を使用すれば、個別のデータベース処理を同時に実行することができます。これらの処理をプロセスと呼びます。

マルチプロセスは、1つのコンピュータ上でのマルチユーザのようなものです。それぞれの処理は個別のタスクです。つまりそれぞれのメソッドを個別のデータベースタスクとして実行することができるということです。

この節では以下の点について説明します:

- プロセスの作成とクリア
- プロセスの要素
- ユーザプロセス
- 4Dが作成するプロセス
- ローカルおよびグローバルプロセス
- プロセス間のレコードロック

Note: この節ではストアドプロシージャを説明していません。この点については4D Server Referenceマニュアルのストアドプロシージャの節を参照してください。

プロセスの作成とクリア

新規プロセスを作成するにはいくつかの方法があります:

- デザインモードにおいて、**メソッド実行**ダイアログボックスで**新規プロセス**チェックボックスをチェックした後、メソッドを実行する。**メソッド実行**ダイアログボックスで選択したメソッドが (そのプロセスをコントロールする) プロセスメソッドとなります。
- メニューを選択してプロセスを開始できます。**メニューエディタ**において、メニューを選択して**新規プロセス開始** チェックボックスをクリックします。メニューコマンドに関連付けられたメソッドがプロセスメソッドです。
- New process関数を使用する方法。New process関数の引数として渡されたメソッドがプロセスメソッドです。
- Execute on server関数を使用して、サーバ上にストアドプロシージャを作成する。関数の引数として渡されたメソッドがプロセスメソッドです。

プロセスは以下の条件でクリアできます。最初の2つは自動的に行われます:

- プロセスメソッドの実行が完了したとき。
- ユーザがデータベースを終了したとき。
- メソッドからプロセスを中止するか、またはデバッガからアボートボタンを使用した場合。
- ランタイムエクスプローラから**アボート**を選択した場合。

プロセスは別のプロセスを作成することができます。プロセスは階層構造にはなっていません。どのプロセスから作成されようと、すべてのプロセスは同等です。いったん、"親"プロセスが"子"プロセスを作成すると、親プロセスの実行状況に関係なく、子プロセスは処理を続行します。

プロセスの要素

各プロセスには個々の要素があります。プロセスには以下の3種類の要素があります:

- **インタフェース要素**: プロセスを表示するのに必要な要素。
- **データ要素**: データベース内のデータに関連する情報。
- **ランゲージ要素**: メソッドで使用される、アプリケーションを開発する際に重要な要素。

インタフェース要素

インタフェース要素には以下のものがあります:

- **メニューバー**: 各プロセスは、独自のカレントメニューバーを持つことができます。最前面のプロセスのメニューバーがデータベースのカレントメニューバーになります。
- **1つ以上のウィンドウ**: 各プロセスは、1つまたは複数のウィンドウを同時に開くことができます。ウィンドウをまったく持たないプロセスもあります。
- **1つのアクティブ (最前面) ウィンドウ**: プロセスは、複数のウィンドウを同時に開くことができますが、アクティブウィンドウは各プロセスに1つしかありません。複数のアクティブウィンドウを持つには、アクティブウィンドウの数だけのプロセスを起動しなければなりません。

Note: サーバ上で実行されるプロセス (ストアドプロシージャ) はインタフェース要素を含んではいけません。

データ要素

データ要素は、データベースが使用するデータを参照します。以下のようなデータ要素があります:

- **テーブルごとのカレントセレクション**: 各プロセスは、個々にカレントセレクションを持ちます。1つのテーブルに対して別々のプロセスにそれぞれのカレントセレクションを持つことができます。
- **テーブルごとのカレントレコード**: 各テーブルは、プロセスごとに異なるカレントレコードを持つことができます。

Note: データ要素に関するこの記述は、プロセスがグローバルスコープの場合に有効です。デフォルトですべてのプロセスはグローバルです。後述のグローバルプロセスとローカルプロセスの段落を参照してください。

ランゲージ要素

プロセスのランゲージ要素は、4Dでのプログラミングに関連した要素です。

- **変数**: すべてのプロセスは独自のプロセス変数を持っています。詳細はの節を参照してください。プロセス変数はその本来のプロセスの範囲内でのみ認識されます。
- **デフォルトテーブル**: プロセスごとに独自のデフォルトテーブルを持っています。
- **入力フォームと出力フォーム**: 各プロセスの各テーブルに対して、デフォルトの入力フォームと出力フォームをメソッドから設定することができます。
- **プロセスセット**: 各プロセスは、独自のプロセスセットを持っています。LockedSetはプロセスセットです。プロセスセットはプロセスメソッドが終了すると直ちに消去されます。
- **プロセスごとのOn Error Call**: 各プロセスは、独自のエラー処理メソッドを持ちます。
- **デバッグウィンドウ**: 各プロセスは、独自のウィンドウを持つことができます。

ユーザプロセス

ユーザプロセスは、特定のタスクを行うために開発者が作成するプロセスです。ユーザプロセスはカーネルプロセスと処理時間を共有します。例えばWeb接続プロセスはユーザプロセスです。

4Dアプリケーションは自身が必要とするプロセスも作成します。以下のプロセスは4Dが作成し管理します:

- **メインプロセス**: メインプロセスはユーザインタフェースのウィンドウ表示を管理します。
- **デザインプロセス**: デザインプロセスはデザインモードのウィンドウとエディタを管理します。インタプリタコードを含まないコンパイルされたデータベースには、デザインプロセスはありません。
- **Webサーバプロセス**: Webサーバプロセスは、Web上にデータベースが公開された時に起動します。これに関する詳細はWebサーバ設定と接続管理の節を参照してください。
- **キャッシュマネージャプロセス**: キャッシュマネージャプロセスはデータベースのディスクI/ O を管理します。このプロセスは4D や 4D Server が実行されるとすぐに開始されます。
- **インデックスプロセス**: インデックスフィールドは、分離されたプロセスとしてデータベースのフィールドのインデックスを管理します。このプロセスはフィールドのインデックスが構築されたり削除されたりする際に作成されます。
- **On Event Managerプロセス**: このプロセスはON EVENT CALL コマンドでイベント処理メソッドがインストールされると、作成されます。このプロセスはイベントが発生すると、ON EVENT CALLでインストールされたイベントメソッドを実行します。イベントメソッドはこのプロセスのプロセスメソッドです。このプロセスは、メソッドが実行されていなくても継続して実行されます。イベント処理はデザインモードでも発生します。

グローバルプロセスとローカルプロセス

プロセスのスコープにはローカルとグローバルがあります。デフォルトですべてのプロセスはグローバルです。

グローバルプロセスはデータへのアクセスや操作を含め、あらゆる処理を実行できます。ほとんどの場合グローバルプロセスを使用します。

ローカルプロセスは、データアクセスを必要としない処理の場合にだけ使用することをお勧めします。例えば、イベント処理メソッドを実行、またはフローティングウィンドウ等のインタフェース要素を制御するためにローカルプロセスを使用します。

名前によってプロセスがローカルであることを指定します。ローカルプロセス名は、先頭にドル記号 ($) を付ける必要があります。

警告: ローカルプロセスでデータアクセスを行った場合、メインプロセスでアクセスすることになり、そのプロセス内で実行される処理との間でコンフリクトが起きるおそれがあります。

4D Server: データアクセスを行わない処理に対し、クライアント側でローカルプロセスを使用すると、サーバの負荷が軽減されます。

プロセス間のレコードロック

他のプロセスが修正のためにレコードを正常にロードすると、そのレコードはロックされます。ロックされているレコードは、別のプロセスでロードすることはできますが、修正することはできません。レコードは、それを修正しているプロセス内でのみロックが解除されます。レコードをロック解除の状態でロードするには、テーブルが"読み書き可能モードになっていなければなりません。詳細はの節を参照してください。

Count tasks

Count tasks -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	開いているプロセスの (カーネルプロセスを含む)

説明

Count tasksは、4Dや4D Server (ストアドプロシージャ) で開かれているプロセスの数を返します。

この数には、4Dが自動的に管理するものも含めてすべてのプロセスが含まれます。この中にはメインプロセス、デザインプロセス、キャッシュマネージャプロセス、インデックスプロセス、およびWebサーバプロセスが含まれます。

Count tasksにより返される数にはアボートされたプロセスも含まれます。

例題

Process stateとの例題参照

Count user processes

Count user processes -> 戻り値

引数	型		説明
戻り値	倍長整数	←	活動中のプロセス (内部プロセスを除く)

説明

Count user processesは4Dアプリケーションで現在活動中のプロセスの数を返します。対象のプロセスにはタイプが -25 (Internal Timer Process), -31 (Client Manager Process) そして -15 (Server Interface Process) のものは含まれません。プロセスタイプに関する詳細はPROCESS PROPERTIES コマンドと定数テーマを参照してください。
Count user processesは、ユーザが直接あるいは間接に開いたプロセスの数を返します (PROCESS PROPERTIESコマンドから返されるorigin引数の値が0以上のプロセス)。

Note: "活動中の" プロセスとは、ステータスがabortedでもdoes not existでもないプロセスです (Process state コマンド参照)。

Count users

Count users -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	サーバに接続しているユーザ数

説明

Count users コマンドは、サーバ上のストアドプロシージャから呼び出された場合、サーバに接続しているユーザの数を返します。

サーバ上で少なくとも1つのストアドプロシージャが実行されていて、Count usersが他のコンテキスト (クライアントマシン、Webメソッド) から呼ばれた場合、コマンドはユーザの数に+1して返します。

シングルユーザの4Dの場合、Count usersは1を返します。

Current process

Current process -> 戻り値

引数	型		説明
戻り値	倍長整数	⮐	プロセス番号

説明

Current processは、このコマンドを呼び出したプロセスのプロセス番号を返します。

例題

DELAY PROCESSとPROCESS PROPERTIESの例題参照

DELAY PROCESS

DELAY PROCESS (process ; duration)

引数	型		説明
process	倍長整数	→	プロセス番号
duration	倍長整数	→	遅延時間 (tick)

説明

DELAY PROCESSはプロセスの実行を指定したtick数 (1tick＝1/60秒) だけ遅らせます。この間、そのprocessは処理時間を使用しません。プロセスの実行を遅延しても、そのプロセスはメモリ内に残ります。

プロセスが既に遅延状態の場合、このコマンドはそれを再度遅延します。この場合、durationは残時間に加算されるのではなく、それに置き換わります。したがって、これ以上遅延させたくなければゼロ (0) を渡します。

プロセスが存在しなければ、このコマンドは何も行いません。

Note: クライアントマシンから、サーバマシンで動作するストアドプロシージャ (process<0) に対して、このコマンドを適用することはできません。

例題 1

の例題参照

例題 2

Process numberの例題参照

EXECUTE ON CLIENT

EXECUTE ON CLIENT (clientName ; methodName {; param}{; param2 ; ... ; paramN})

clientName	文字	⇨	4D Clientの登録名
methodName	文字	⇨	実行するメソッドの名前
param		⇨	メソッドの引数

説明

EXECUTE ON CLIENT コマンドは、clientNameという名前で登録されている4D Clientで、必要であればparam1... paramNを引数とし、methodNameメソッドを実行します。4D Clientの登録名はREGISTER CLIENT コマンドで定義します。
このコマンドは、4D Clientまたは4D Serverのストアドプロシージャで呼び出すことができます。

メソッドが1つ以上の引数を要求する場合、メソッドの名前の後に引数を渡します。
4D Client上でのメソッド実行は、クライアントワークステーション上で自動的に作成されたプロセス内で行われます。そのプロセス名は4D Clientの登録名です。

このコマンドが同じ4D Clientに対して連続的に呼び出されると、実行がスタックされます。つまりメソッドは非同期モードで、次から次へと処理されます。スタックされたメソッドが増えると、4D Clientのワークロードも大きくなってしまいます。GET REGISTERED CLIENTSコマンドを使用して、各クライアントのワークロードの状態を知ることができます。

Note: 実行順序のスタックは、UNREGISTER CLIENTコマンドを使って4D Clientが登録解除されない限りは、変更することも止めることもできません。

登録された複数の4D Clientに対し、同じメソッドを同時に実行させる事ができます。これを実行するには、clientName引数にワイルドカード (@) を使用します。

OKシステム変数は、4D Serverがメソッドの実行要求を正しく受け取った場合、1になりますが、これはメソッドが4D Clientによって正しく実行されたということを保証するものではありません。

例題 1

"GenerateNums"メソッドを、"Client1"クライアント上で実行したいと仮定します:

```
EXECUTE ON CLIENT("Client1";"GenerateNums";12;$a;"Text")
```

例題 2

すべてのクライアントに"EmptyTemp"メソッドを実行させたい場合には:

```
EXECUTE ON CLIENT("@";"EmptyTemp")
```

例題 3

REGISTER CLIENT コマンドの例題参照

システム変数およびセット

4D Serverが正しく実行リクエストを受け取ると、OKシステム変数は1に設定されます。しかしこのことは4Dクライアント上でメソッドが正しく実行されることを保証するものではありません。

Execute on server

Execute on server (procedure ; stack {; name {; param {; param2 ; ... ; paramN}}}{; *}) -> 戻り値

引数	型		説明
procedure	文字	→	プロセス中で実行するメソッド
stack	倍長整数	→	スタックサイズ (バイト)
name	文字	→	作成するプロセスの名前
param	式	→	メソッドの引数
*	演算子	→	重複しないプロセス
戻り値	倍長整数	↩	新規プロセスのプロセス番号 または既存のプロセス番号

説明

Execute on server コマンドはサーバマシン上 (クライアント/サーバで実行された場合) 、または同じマシン上 (シングルユーザで実行された場合) で新しいプロセスを開始し、そのプロセスのプロセス番号を返します。

Execute on serverコマンドを使用してストアドプロシージャを開始します。ストアドプロシージャについての詳細は4D Server Referenceマニュアルのストアドプロシージャの節を参照してください。

クライアントマシンでExecute on serverを実行した場合、負のプロセス番号が返されます。サーバマシンでExecute on serverを実行すると、正のプロセス参照番号が返されます。サーバマシン上でNew processコマンドを実行することは、Execute on serverを実行することと同じです。

プロセスが作成できない場合 (例えばメモリ不足) 、Execute on serverは0を返し、エラーが発生します。このエラーはON ERR CALLでインストールしたエラー処理メソッドを使用してとらえることができます。

プロセスメソッド

procedureには、新しいプロセスのプロセスメソッド名を指定します。4Dは新規プロセスのコンテキストを設定した後、このメソッドの実行を開始します。したがって、これがプロセスメソッドになります。

プロセススタック

stackには、プロセスのスタックに割り当てるメモリの量を指定します。このメモリ領域にメソッド呼び出し、ローカル変数、サブルーチンの引数、スタックしたレコード等が"積み上げ"られてゆきます。この大きさはバイトで表され、少なくとも64K (約64,000バイト) を渡します。しかし、プロセスでサイズの大きいコールを続けて実行する場合等は (サブルーチンからサブルーチンを呼び出す等) 、これ以上の値を渡すこともできます。例えば、必要なら200K (約200,000バイト) を渡すこともできます。

Note: スタックはプロセスの合計メモリではありません。各プロセスはレコードやインタープロセス変数等のためにメモリを共有します。またプロセスはプロセス変数の保持にメモリを余計に使用します。スタックはローカル変数やメソッドコール、サブルーチンの引数、スタックされたレコードを維持するだけです。

64-bit 4D Serverに関する注意: 64-bit 4D Serverで実行されるプロセスのスタックは32-bit 4D Serverより多くのメモリー量を必要とします (大体2倍程度)。このことから最低でも128,000バイト、連鎖呼び出しがある場合は一般的に400,000バイト程度のスタックサイズを渡すことが 推奨されます。64-bit 4D Serverでコードが実行されることが予測される場合、この引数の値をチェックしてください。

プロセス名

nameには新しいプロセスの名前を指定します。シングルユーザモードでは、ここで指定した名前が短タイムエクスプローラのプロセスリストに表示され、この新しいプロセスに対してPROCESS PROPERTIESコマンドを実行するとこの名前が返されます。クライアント/サーバモードでは、4D Serverのメインウインドウのストアドプロシージャリストに青字で表示されます。

この引数は省略することができます。省略した場合、プロセス名は空の文字列になります。

警告: New Processとは異なり、Execute on serverコマンドの実行時に、プロセス名の先頭にドル記号 ($) 記号を付けて、プロセスをローカルにしようとはしないでください。シングルユーザモードではExecute on serverコマンドはNew Processコマンドと同じ処理を実行するため、プロセスをローカルにしても正常に動作します。しかし、クライアント/サーバモードではエラーが発生します。

プロセスメソッドの引数

プロセスメソッドに引数を渡せるようになりました。サブルーチンにパラメータを渡すのと同じ要領でプロセスメソッドに引数を渡します。しかし制約があります。ポインタ表現は渡すことができません。また、メソッドに対して配列を引数として受け渡すことができない点にも留意してください。プロセスメソッドは、新規プロセスのコンテキスト内で実行を開始する際に、$1, $2等に引数の値を受け取ります。

Note: プロセスメソッドに引数を渡す場合、必ずname引数を指定しなければなりません。この場合、この引数は省略できません。

param または戻り値に 4Dオブジェクト(C_OBJECT) を渡す場合、サーバーに対してはUTF-8のJSONフォームが使用されます。もし C_OBJECT オブジェクトにポインターが格納されている場合、そのポインターが参照している値が送られ、ポインターそのものは送られる訳ではないことに注意して下さい。

オプションの * 引数

この最後の引数を指定した場合、4Dははじめにnameに指定した名前を持つプロセスが既に実行されているかどうかを調べます。同一名のプロセスが存在する場合、4Dは新規プロセスを開始せずにその名前を持つプロセスのプロセス番号を返します。

例題

以下の例は、クライアント/サーバにおいてデータ読み込みの処理速度を飛躍的に向上する方法を示しています。以下のRegular Importメソッドにより、クライアント側でIMPORT TEXTコマンドを使用したレコード読み込みに要する時間を調べることができます:

```
  ` Regular Import Project Method
 $vhDocRef:=Open document("")
 If(OK=1)
    CLOSE DOCUMENT($vhDocRef)
    FORM SET INPUT([Table1];"Import")
    $vhStartTime:=Current time
    IMPORT TEXT([Table1];Document)
    $vhEndTime:=Current time
    ALERT("It took "+String(0+($vhEndTime-$vhStartTime))+" seconds.")
 End if
```

通常のデータ読み込みでは、4D Clientがテキストファイルの解析を行った後、各レコードごとに新規レコードを作成し、読み込んだデータをフィールドに格納して、データベースに追加 するためにレコードをサーバマシンに送信します。この結果、ネットワーク上には大量のリクエストが行きかいます。この処理を最適化するには、ストアドプロ シージャを利用し、サーバマシン上でローカルにジョブを実行します。クライアントマシンではドキュメントファイルをBLOBにロードし、引数にこの BLOBを渡してストアドプロシージャを開始します。ストアドプロシージャはこのドキュメントファイルをサーバマシンに保存し、ドキュメントをローカルに 読み込みます。したがって、ネットワークリクエストの大部分が必要なくなるため、データ読み込みはシングルユーザ版並の速度でローカルに実行されます。次 にCLIENT IMPORTプロジェクトメソッドを示します。このメソッドはクライアントマシン上で実行され、後のSERVER IMPORTストアドプロシジャを開始します。

```
  ` CLIENT IMPORT プロジェクトメソッド
  ` CLIENT IMPORT ( ポインタ; 文字列 )
  ` CLIENT IMPORT ( -> [table] ; Input form )

 C_POINTER($1)
 C_STRING(31;$2)
 C_TIME($vhDocRef)
 C_BLOB($vxData)
 C_LONGINT(spErrCode)

  ` 読み込むドキュメントを選択
 $vhDocRef:=Open document("")
 If(OK=1)
  ` ドキュメントが選択されたら、開いておく必要はない
    CLOSE DOCUMENT($vhDocRef)
    $vhStartTime:=Current time
  ` ドキュメントをメモリにロード
    DOCUMENT TO BLOB(Document;$vxData)
    If(OK=1)
  ` ドキュメントがBLOBに読み込めたら
  ` サーバマシン上で読み込みを実行するストアドプロシージャを開始
       $spProcessID:=Execute on server("SERVER IMPORT";32*1024;
       "Server Import Services";Table($1);$2;$vxData)
  ` この時点でBLOBはもう必要ない
       CLEAR VARIABLE($vxData)
  ` ストアドプロシージャの処理を待つ
       Repeat
          DELAY PROCESS(Current process;300)
          GET PROCESS VARIABLE($spProcessID;spErrCode;spErrCode)
          If(Undefined(spErrCode))
  ` Note: ストアドプロシージャが自身の変数spErrCodeインスタンスをまだ初期化していない場合、
  ` 未定義変数が返されることがある
             spErrCode:=1
          End if
       Until(spErrCode<=0)
  ` ストアドプロシージャに結果を受け取ったことを通知
       spErrCode:=1
       SET PROCESS VARIABLE($spProcessID;spErrCode;spErrCode)
       $vhEndTime:=Current time
       ALERT("It took "+String(0+($vhEndTime-$vhStartTime))+" seconds.")
    Else
       ALERT("There is not enough memory to load the document.")
    End if
 End if
```

次に、ストアドプロシージャとして実行されるSERVER IMPORTプロジェクトメソッドを示します:

```
  ` SERVER IMPORT プロジェクトメソッド
  ` SERVER IMPORT ( 倍長整数; 文字列 ; BLOB )
  ` SERVER IMPORT ( テーブル番号 ; 入力フォーム ; Iデータの入力 )

 C_LONGINT($1)
 C_STRING(31;$2)
 C_BLOB($3)
 C_LONGINT(spErrCode)

  ` Operation is not finished yet, set spErrCode to 1
 spErrCode:=1
 $vpTable:=Table($1)
 FORM SET INPUT($vpTable->;$2)
 $vsDocName:="Import File "+String(1+Random)
 DELETE DOCUMENT($vsDocName)
 BLOB TO DOCUMENT($vsDocName;$3)
 IMPORT TEXT($vpTable->;$vsDocName)
 DELETE DOCUMENT($vsDocName)
  ` Operation is finished, set spErrCode to 0
 spErrCode:=0
  ` Wait until the requester Client got the result back
 Repeat
    DELAY PROCESS(Current process;1)
 Until(spErrCode>0)
```

データベースにこの2つのプロジェクトメソッドを作成したら、例えば 以下のようにストアドプロシージャベースのデータ読み込みを実行できます:

```
 CLIENT IMPORT(->[Table1];"Import")
```

ベンチマークテストによると、このメソッドを使用してレコード読み込みを実行すると通常の読み込みよりも60倍高速に処理されました。

GET REGISTERED CLIENTS

GET REGISTERED CLIENTS (clientList ; methods)

引数	型	説明
clientList	テキスト配列	⇐ 登録されている4D Clientのリスト
methods	倍長整数配列	⇐ 実行するメソッドのリスト

説明

GET REGISTERED CLIENTS コマンドは2つの配列を作成します:

- clientListsは、REGISTER CLIENTコマンドを使って"登録された"クライアントの登録名の配列となります。
- methodsは、各クライアントの"ワークロード"の配列となります。ワークロードとは、EXECUTE ON CLIENT コマンドでスタックされた、4D Clientがこれから実行しなければならないメソッドの数です (より詳しい情報は、EXECUTE ON CLIENTコマンドを参照してください)。

Note: 処理に成功すると、OKシステム変数は1になります。

例題 1

すべての登録されたクライアントの配列と、まだ実行されずに残っているメソッド数の配列を取得します:

```
ARRAY TEXT($clients;0)
ARRAY LONGINT($methods;0)
GET REGISTERED CLIENTS($clients;$methods)
```

例題 2

REGISTER CLIENT コマンドの例題参照

システム変数およびセット

処理が成功すると、システム変数OKは1に設定されます。

New process

New process (method ; stack {; name {; param {; param2 ; ... ; paramN}}}{; *}) -> 戻り値

method	文字	⇨	プロセスで実行させるメソッド
stack	倍長整数	⇨	スタックサイズ (バイト)
name	文字	⇨	作成するプロセスの名前
param	式	⇨	メソッドに渡す引数
*		⇨	重複しないプロセス

説明

New process コマンドは、(同じマシン上で) 新しいプロセスを開始し、そのプロセス参照番号を返します。

プロセスが作成できない場合 (例えば、メモリ不足) 、New processは0を返し、エラーが発生します。このエラーはON ERR CALLでインストールされたエラー処理メソッドを使用してとらえることができます。

プロセスメソッド

methodには、新しいプロセスのプロセスメソッド名を指定します。4Dは新規プロセスのコンテキストを設定した後、このメソッドの実行を開始します。したがって、これがプロセスメソッドになります。

プロセススタック

stackには、プロセスのスタックに割り当てるメモリの量を指定します。このメモリ領域にメソッド呼び出し、ローカル変数、サブルーチンの引数、スタックしたレコード等が"積み上げ"られてゆきます。この大きさはバイトで表され、少なくとも64K (約64,000バイト) を渡します。しかし、プロセスでサイズの大きいコールを続けて実行する場合等は (サブルーチンからサブルーチンを呼び出す等) 、これ以上の値を渡すこともできます。例えば、必要なら200K (約200,000バイト) を渡すこともできます。

Note: スタックはプロセスの合計メモリではありません。各プロセスはレコードやインタープロセス変数等のためにメモリを共有します。またプロセスはプロセス変数の保持に追加のメモリを使用します。スタックにはさまざまな4Dの情報が格納されます。スタックに格納される情報の量は、ネストしたメソッドの呼び出し数、前のフォームが閉じられる前に開かれたフォームの数、ネストしたメソッドで使用されるローカル変数のサイズに基づきます。

64-bit 4D Serverに関する注意: 64-bit 4D Serverで実行されるプロセスのスタックは32-bit 4D Serverより多くのメモリー量を必要とします (大体2倍程度)。このことから最低でも128,000バイト、連鎖呼び出しがある場合は一般的に400,000バイト程度のスタックサイズを渡すことが推奨されます。64-bit 4D Serverでコードが実行されることが予測される場合、この引数の値をチェックしてください。

プロセス名

nameには新しいプロセスの名前を指定します。シングルユーザモードでは、ここで指定した名前が短タイムエクスプローラのプロセスリストに表示され、この新しいプロセスに対してPROCESS PROPERTIESコマンドを実行するとこの名前が返されます。この引数は省略することができます。省略した場合、プロセス名は空の文字列になります。ローカルスコープのプロセスを作成するには、名前の先頭にドルサイン ($) をつけます。

重要: クライアント/サーバにおいて、ローカルプロセスはデータにアクセスしてはいけないことに注意してください。

プロセスメソッドの引数

バージョン6より、プロセスメソッドに引数を渡せるようになりました。サブルーチンにパラメータを渡すのと同じ要領でプロセスメソッドに引数を渡します。しかし制約があります。ポインタ表現は渡すことができません。また、メソッドに対して配列を引数として受け渡すことができない点にも留意してください。プロセスメソッドは、新規プロセスのコンテキスト内で実行を開始する際に、$1, $2等に引数の値を受け取ります。

Note: プロセスメソッドに引数を渡す場合、必ずname引数を指定しなければなりません。この場合、この引数は省略できません。

オプションの * 引数

この最後の引数を指定した場合、4Dははじめにnameに指定した名前を持つプロセスが既に実行されているかどうかを調べます。同一名のプロセスが存在する場合、4Dは新規プロセスを開始せずにその名前を持つプロセスのプロセス番号を返します。

例題

以下のプロジェクトメソッドがある時:

```
  ` ADD CUSTOMERS
 SET MENU BAR(1)
 Repeat
    ADD RECORD([Customers];*)
 Until(OK=0)
```

メニューバーエディタでカスタムメニュー項目にこのプロジェクトメソッドを指定し、**新規プロセス開始** チェックボックスをチェックしている場合、4Dはそのメソッドを実行する新規プロセスを自動的に開始します。SET MENU BAR(1)を実行すると、この新規プロセスに対してメニューバーが追加されます。ウィンドウ (Open windowコマンドでオープンするウインドウ) が何も存在しない場合、ADD RECORDコマンドを実行することにより、自動的にウィンドウが開かれます。

カスタムのコントロールパネルのボタンをクリックすると、"Add Customers"プロセスが開始されるようにするには、以下のようにします:

```
  ` bAddCustomers button object method
 $vlProcessID:=New process("Add Customers";32*1024;"Adding Customers")
```

このボタンは、先に作ったメニュー項目と同じことを行います。

メニュー項目を選択したり、このボタンをクリックすると、プロセスを開始 (そのプロセスが存在しない場合) またはプロセスを前面に配置 (そのプロセスが既に実行中の場合) したい場合、以下のようにSTART ADD CUSTOMERSメソッドを作成することができます:

```
  ` START ADD CUSTOMERS
 $vlProcessID:=New process("Add Customers";64*1024;"Adding Customers";*)
 If($vlProcessID#0)
    BRING TO FRONT($vlProcessID)
 End if
```

bAddCustomers のオブジェクトメソッドは以下のようになります:

```
  ` bAddCustomers button object method
 START ADD CUSTOMERS
```

メニューバーエディタでADD CUSTOMERSメソッドをSTART ADD CUSTOMERSメソッドと置き換え、メニュー項目の**新規プロセス開始** チェックボックスを選択解除します。

PAUSE PROCESS

PAUSE PROCESS (process)

引数	型	説明
process	倍長整数	→ プロセス番号

説明

PAUSE PROCESSは、RESUME PROCESSコマンドで再開されるまでprocessの実行を停止します。この間processはマシンの処理時間を使用しません。プロセスは停止されてもメモリ内に残ります。

processが既に停止していた場合は、PAUSE PROCESSコマンドは何も行いません。プロセスがDELAY PROCESSコマンドで遅延されている場合もそのプロセスは停止されます。RESUME PROCESSコマンドは、即座にプロセスを再開します。

プロセスの実行を停止している間はそのプロセスのウィンドウに入力することはできません。この場合には、ユーザの混乱を避けるためにそのプロセスを非表示にするとよいでしょう。processが存在しなければ、このコマンドは何も行いません。

警告: PAUSE PROCESSコマンドは、あなたが開始したプロセス内だけで使用してください。PAUSE PROCESSコマンドはメインプロセスに何も影響を与えません。

Note: クライアントマシンから、サーバーマシンで動作するストアドプロシージャ (process<0) に対して、このコマンドを使用してはいけません。

Process aborted

Process aborted -> 戻り値

引数	型		説明
戻り値	ブール	⮐	True = プロセスは中止されようとしている, False = プロセス中止されようとしていない

説明

Process abortedコマンドは、このコマンドを呼び出したプロセスが不意に中断されようとしている場合**True**を返します。これはコマンドの実行が正常に完了できないことを意味します。例えばこれはQUIT 4Dを呼び出した後に発生します。

Process number

Process number (name {; *}) -> 戻り値
name 文字→プロセス番号を取り出すプロセス名
* →4D Serverのプロセス番号を返す

説明

Process numberは、name引数に指定した名前を持つプロセス番号を返します。プロセスが見つからない場合には、Process numberは0を返します。

オプションの引数 * を指定すると、サーバ上で実行されたプロセス（ストアドプロシージャ）のプロセスIDを4D Clientで取得することができます。この場合、負の値が返されます。GET PROCESS VARIABLEやSET PROCESS VARIABLEコマンドを使用する際には、このオプションが特に役立ちます。詳細はそれぞれのコマンドの説明を参照してください。

サーバマシン上のプロセスから引数 * を指定してこのコマンドを実行すると、正の値が返されます。

例題

独立したプロセスで実行するカスタムフローティングウインドウを作成します。このプロセスでは、デザインモードでやり取りができる独自のツールを実装します。例えば、キーワードの階層リストで項目を選択すると、デザインモードの最前面ウインドウにテキストを貼り付けるようなツール。これを実行するにはペーストボードを使用できますが、貼り付けイベントはデザインプロセスの内部で発生する必要があります。以下の関数は、デザインプロセス（が実行している場合）のプロセス番号を返します:

```
  ` Design process number プロジェクトメソッド
  ` Design process number -> 倍長整数
  ` Design process number -> デザインプロセス番号

 $0:=Process number("デザインプロセス")
  ` Note: プロセス名が変更されると、このコードは利用できなくなります
```

この関数を使用して、下記のプロジェクトメソッドは、引数として受け取ったテキストをデザインモードの最前面のウインドウに貼り付けます（適用可能な場合）:

```
  ` PASTE TEXT TO DESIGN プロジェクトメソッド
  ` PASTE TEXT TO DESIGN ( テキスト)
  ` PASTE TEXT TO DESIGN ( デザインモードの最前面ウィンドウに張り付けるテキスト)

 C_TEXT($1)
 C_LONGINT($vlDesignPID;$vlCount)

 $vlDesignPID:=Design process number
 If($vlDesignPID #0)
  ` ペーストボードにテキストを置く
   SET TEXT TO PASTEBOARD($1)
  ` Ctrl-V / Cmd-V イベントをポスト
   POST KEY(Character code("v");Command key mask;$vlDesignPID)
  ` DELAY PROCESSを繰り返し呼び出して、スケジューラに
  ` デザインプロセスへのイベントを渡すチャンスを与える
   For($vlCount;1;5)
      DELAY PROCESS(Current process;1)
   End for
 End if
```

PROCESS PROPERTIES

PROCESS PROPERTIES (process ; procName ; procState ; procTime {; procVisible {; uniqueID {; origin}}})

引数	型		説明
process	倍長整数	→	プロセス番号
procName	文字	←	プロセス名
procState	倍長整数	←	プロセスの状態
procTime	倍長整数	←	プロセスの稼働時間 (Tick)
procVisible	ブール	←	TRUE: 表示, FALSE: 非表示
uniqueID	整数	←	ユニークなプロセス番号
origin	倍長整数	←	プロセスの発生源

説明

PROCESS PROPERTIESコマンドはprocessに渡したプロセス番号を持つプロセスに関する情報を返します。

このコマンド呼出し後に:

- procNameはプロセスの名前を返します。プロセス名で注意すべき点は、次のとおりです:
  - プロセスがメソッド実行ダイアログボックスから (新規プロセスオプションを選択した状態で) 起動された場合、プロセスの名前は"**P_**"で始まりその後に数字が続きます。
  - プロセス開始プロパティが選択されたカスタムメニューから起動されたプロセスの場合には、プロセスの名前は"**M_**"または"**ML_**"で始まり、その後に数字が続きます。
  - Webサーバーによりプロセスが開始されれた場合、名前は"Web Process#"+UUIDです。
  - プロセスがアボートしていた場合（そして、そのスロットがまだ再使用されていない場合)、プロセスの名前はそのまま返されます。プロセスがアボートしたかどうかを検出するには、procState=-1を判定します（下記の表を参照）。

- procStateは、呼び出しを行った時点のプロセスの状態を返します。この引数は次の定義済定数で提供される値のいずれか1つを返します:

定数	型	値
Aborted	倍長整数	-1
Delayed	倍長整数	1
Does not exist	倍長整数	-100
Executing	倍長整数	0
Hidden modal dialog	倍長整数	6
Paused	倍長整数	5
Waiting for input output	倍長整数	3
Waiting for internal flag	倍長整数	4
Waiting for user event	倍長整数	2

- procTimeは、プロセスが起動されてからの稼働時間をtick数 (1/60秒) で返します。
- procVisibleが指定された場合には、プロセスが表示されている場合はTRUE、隠されている場合はFALSEを返します。
- uniqueIDが指定された場合には、独自のプロセス番号を返します。各プロセスはプロセス番号と、セッション内でユニークなプロセス番号を持っています。ユニークプロセス番号は、2つのプロセス間または2つのプロセスセッションの間で区別することができるようにするものです。これは4Dセッションの間に開始されたプロセス番号に対応するものです。
- originが指定された場合には、プロセスの発生元を表わす値を返します。4Dは下記の定数を定義しています (Process Typeテーマ内):

定数	型	値
_o_Web Process with Context	倍長整数	-11
Apple Event Manager	倍長整数	-7
Backup Process	倍長整数	-19
Cache Manager	倍長整数	-4
Client Manager Process	倍長整数	-31
Created from execution dialog	倍長整数	3
Created from Menu Command	倍長整数	2
Design Process	倍長整数	-2
Event Manager	倍長整数	-8
Execute on Client Process	倍長整数	-14
Execute on Server Process	倍長整数	1
External Task	倍長整数	-9
Indexing Process	倍長整数	-5
Internal 4D Server Process	倍長整数	-18
Internal Timer Process	倍長整数	-25
Log File Process	倍長整数	-20
Main Process	倍長整数	-1
Method editor macro Process	倍長整数	-17
Monitor Process	倍長整数	-26
MSC Process	倍長整数	-22
None	倍長整数	0
On Exit Process	倍長整数	-16
Other 4D Process	倍長整数	-10
Other User Process	倍長整数	4
Process Server Interface	倍長整数	-15
Restore Process	倍長整数	-21
Serial Port Manager	倍長整数	-6
SQL Method Execution Process	倍長整数	-24
Web Process on 4D Remote	倍長整数	-12
Web Process with no Context	倍長整数	-3
Web server Process	倍長整数	-13

注: 4Dの内部プロセスは負の値を返し、ユーザが作成したプロセスは正の値を返します。

プロセスが存在しない場合、つまり1からCount tasksまでの番号を渡さなかった場合、**PROCESS PROPERTIES**コマンドは変数パラメータの変更を行いません。

例題 1

次の例は、変数vName, vState, vTimeSpentに現在のプロセスのプロセス名、プロセスステータス、プロセス時間を返します:

```
C_TEXT(vName) ` Initialize the variables
C_LONGINT(vState)
C_LONGINT(vTime)
PROCESS PROPERTIES(Current process;vName;vState;vTimeSpent)
```

例題 2

On Exitデータベースメソッドの例題参照

Process state

Process state (process) -> 戻り値

引数	型		説明
process	倍長整数	→	プロセス番号
戻り値	倍長整数	↩	プロセスの状態

説明

Process state コマンドは、processに指定したプロセス番号を持つプロセスの状態を返します。

プロセスのステータスとしては以下のような定数があらかじめ定義されています:

定数	型	値
Aborted	倍長整数	-1
Delayed	倍長整数	1
Does not exist	倍長整数	-100
Executing	倍長整数	0
Hidden modal dialog	倍長整数	6
Paused	倍長整数	5
Waiting for input output	倍長整数	3
Waiting for internal flag	倍長整数	4
Waiting for user event	倍長整数	2

プロセスが存在しない (つまり1からCount tasksまでの番号を渡さなかった) 場合、Process stateはDoes not exist (-100)を返します。

例題

以下の例は、各プロセスの名前とプロセス参照番号を配列asProcNameとaiProcNumに入れます。このメソッドは、プロセスがアボートされたかを調べます。この場合、プロセス名とプロセス番号は配列に追加されません:

```
$vlNbTasks:=Count tasks
ARRAY STRING(31;asProcName;$vlNbTasks)
ARRAY INTEGER(aiProcNum;$vlNbTasks)
$vlActualCount:=0
For($vlProcess;1;$vlNbTasks)
   If(Process state($vlProcess)>=Executing)
      $vlActualCount:=$vlActualCount+1
      PROCESS PROPERTIES($vlProcess;asProcName{$vlActualCount};$vlState;$vlTime)
      aiProcNum{$vlActualCount}:=$vlProcess
   End if
End for
  ` Eliminate unused extra elements
ARRAY STRING(31;asProcName;$vlActualCount)
ARRAY INTEGER(aiProcNum;$vlActualCount)
```

REGISTER CLIENT

REGISTER CLIENT (clientName {; period}{; *})

引数	型		説明
clientName	文字	→	4Dクライアントセッション名
period	倍長整数	→	***バージョン11.3より無効***
*	演算子	→	ローカルプロセス

説明

REGISTER CLIENT コマンドは、4Dクライアントステーションを、clientNameで指定した名前で4D Serverに登録し、他のクライアントもしくは4D Server (ストアドプロシージャから) が登録されたマシン上で、EXECUTE ON CLIENTコマンドを使ってメソッドを実行できるようにします。一旦登録されると、4Dクライアントは他のクライアント用に1つまたはそれ以上のメソッドを実行することができます。

Notes:

- データベース環境設定ダイアログの、起動時にクライアント登録オプションを使って、4D Serverに接続するクライアントステーションを自動的に登録することができます。
- ローカルモードの4Dでこのコマンドが使用されても効果はありません。
- ひとつ以上の4Dクライアントが同じ登録名を持つことができます。

このコマンドが実行されると、クライアントステーション上にclientNameという名のプロセスが作成されます。このプロセスはUNREGISTER CLIENTコマンドによってのみアボート可能です。
オプションの * 引数を渡すと、作成されるプロセスはローカルプロセスになり、4Dは自動的にプロセス名の始めにドルマーク ($) を付け加えます。そうでない場合は、グローバルプロセスです。

4Dバージョン11.3より、サーバ/クライアント通信のメカニズムが最適化されました。サーバは必要に応じて直接、登録されたクライアントに実行 リクエストを送信します ("プッシュ"テクノロジ)。以前の、クライアントが定期的にサーバに問い合わせする方法は使用されません。period引数は無視されます。

一度コマンドが実行されると、4Dクライアント名を動的に変更することはできません。これを実行するには、UNREGISTERCLIENTコマンドを呼び出し、再度REGISTER CLIENTコマンドを呼び出します。

例題

以下の例題では小さなメッセージングシステムを作成し、クライアントワークステーション間の通信を可能にします。

1) このRegistrationメソッドは4Dクライアントを登録して、他の4Dクライアントからのメッセージを受け取ることができるようにします:

```
  `他の名前で登録する前に登録解除する必要がある
 UNREGISTER CLIENT
 REPEAT
    vPseudoName:=Request("名前を入力:";"ユーザ";"OK";"キャンセル")
 Until((OK=0)|(vPseudoName#""))
 If(OK=0)
    ... ` 何も行わない
 Else
    REGISTER CLIENT(vPseudoName)
 End if
```

2) 以下の指示は、登録されたクライアントのリストを得ることができるようにするものです。これは内に置くことができます:

```
 PrClientList:=New process("4D Client List";32000;"List of registered clients")
```

3) 以下の4D Client Listメソッドは、メッセージ受信可能な登録済み全4Dクライアントの登録名リストを入手します:

```
 If(Application type=4D Remote Mode)
  ` 以下のコードはクライアントサーバでのみ 有効
    $Ref:=Open window(100;100;300;400;-(Palette window+Has window title);"List of registered clients")
    REPEAT
       GET REGISTERED CLIENTS($ClientList;$ListeCharge)
  `$ClientListに登録済みクライアントリストを取得
       ERASE WINDOW($Ref)
       GOTO XY(0;0)
       For($p;1;Size of array($ClientList))
          MESSAGE($ClientList{$p}+Char(Carriage return))
       End for
  `毎秒ごとに表示
       DELAY PROCESS(Current process;60)
    Until(False) ` 無限ループ
 End if
```

4) 下記のメソッドは、登録済みの他の4Dクライアントにメッセージを送ります。これは、送られた4DクライアントでDisplay_Messageメソッドを呼び出します（下記参照）。

```
 $Addressee:=Request("メッセージの宛先:";"")
  ` On Startup データベースメソッドで取得した、メッセージ受信可能者リストの名前を指定
 If(OK#0)
    $Message:=Request("Message:") ` message
    If(OK#0)
       EXECUTE ON CLIENT($Addressee;"Display_Message";$Message) ` メッセージ送信
    End if
 End if
```

5) 以下は、Display_Messageメソッドです:

```
 C_TEXT($1)
 ALERT($1)
```

6) 最後に、以下のメソッドはクライアントステーションが他の4Dクライアントから見えず、メッセージも受け取れなくなるようにします:

```
 UNREGISTER CLIENT
```

システム変数およびセット

4Dクライアントが正しく登録されるとOKシステム変数に1が設定されます。4Dクライアントが既に登録されている場合、コマンドはなにも行わずOKは0に設定されます。

RESUME PROCESS

RESUME PROCESS (process)

引数	型		説明
process	倍長整数	→	プロセス番号

説明

RESUME PROCESSは、実行が停止または遅延されているprocessを再開します。processが停止も遅延もされていない場合、RESUME PROCESSは何も行いません。

processが事前に遅延されてる場合については、PAUSE PROCESSコマンドまたはDELAY PROCESSコマンドを参照してください。プロセスが存在しない場合、このコマンドは何も行いません。

Note: クライアントマシンから、サーバーマシンで動作するストアドプロシージャ(process<0)に対して、このコマンドを使用してはいけません。

UNREGISTER CLIENT

UNREGISTER CLIENT
このコマンドは引数を必要としません

説明

UNREGISTER CLIENT コマンドは、クライアントステーションの登録を解除します。クライアントはREGISTER CLIENTコマンドによって既に登録されているものでなければなりません。

Note: 4Dクライアントは、ユーザがアプリケーションを終了すると自動的に登録を解除します。

4Dクライアントが前もって登録されていなかったり、コマンドをシングルユーザの4Dで実行しても、コマンドは何も行ないません。

クライアントの登録が正しく解除されるとシステム変数OKは1になり、クライアントが登録されていないとOKは0になります。

例題

REGISTER CLIENT コマンドの例題参照

システム変数およびセット

クライアントが正しく登録解除されるとOKシステム変数に1が、クライアントが登録されていなければ0が設定されます。

プロセス (コミュニケーション)

- CALL PROCESS
- CLEAR SEMAPHORE
- GET PROCESS VARIABLE
- Semaphore
- SET PROCESS VARIABLE
- Test semaphore
- VARIABLE TO VARIABLE

CALL PROCESS

CALL PROCESS (process)

引数	型		説明
process	倍長整数	→	プロセス番号

説明

CALL PROCESSは、processの最前面のウィンドウに表示されたフォームを呼び出します。

重要: CALL PROCESSは、同一マシン上で実行されたプロセス間でのみ有効です。

存在しないプロセスを呼び出した場合には、何も行いません。

process (目的のプロセス) で現在フォームが表示されていない場合には何も行いません。目的のプロセスで表示されているフォームがOn Outside callイベントを受け取ります。デザインモードのフォームプロパティウィンドウにおいて、このフォームのOn Outside callイベントを必ず有効にし、フォームメソッドでこのイベントを管理する必要があります。このイベントが無効であったり、またはフォームメソッドでイベントの管理を行わない場合、何も行われません。

Note: On Outside callイベントは、受け取り側である入力フォームの入力状況を変更します。特に、フィールドが編集中である場合には、On Data changeイベントが生成されます。

呼び出し元プロセス (CALL PROCESSが実行されたプロセス) は"待機"しません。CALL PROCESSは即座に効力を持ちます。必要であれば、この目的のために使用する、(GET PROCESS VARIABLEとSET PROCESS VARIABLEにより) 2つのプロセス間で読み書きが可能なインタープロセス変数やプロセス変数を使用して、呼び出したプロセスからの返答を待つループを書かなければなりません。

フォームを表示しないプロセスの間で通信を行うには、GET PROCESS VARIABLEおよびSET PROCESS VARIABLEコマンドを使用してください。

CALL PROCESSにはCALL PROCESS**(-1)**というもう一つの構文があります。

メソッドの実行速度が遅くならないように、4Dはインタープロセス変数が変更されるたびに再描画することはしません。プロセス参照番号の代わりに-1をprocess引数に渡すと、4Dはプロセスを呼び出さず、その代わりに、同一マシン上で実行されているプロセス内のすべてのウィンドウに表示されているインタプロセス変数をすべて更新します。

例題

の例題参照

CLEAR SEMAPHORE

CLEAR SEMAPHORE (semaphore)

引数	型		説明
semaphore	文字	→	クリアするセマフォ

説明

CLEAR SEMAPHOREは、Semaphoreコマンドで設定されたsemaphoreを消去します。

ルールとして、作成されたすべてのセマフォは消去するべきです。セマフォが消去されない場合、セマフォを作成したプロセスが終了するまで、作成されたセマフォはメモリ上に残ります。プロセスは自身が作成したセマフォしか消去することはできません。セマフォを作成していないプロセス内からセマフォを消去しようとしても、何も行いません。

例題

Semaphoreの例題参照

GET PROCESS VARIABLE

GET PROCESS VARIABLE (process ; srcVar ; dstVar {; srcVar2 ; dstVar2 ; ... ; srcVarN ; dstVarN})

引数	型		説明
process	倍長整数	→	ソースプロセス番号
srcVar	変数	→	ソース変数
dstVar	変数	←	受け取る変数

説明

GET PROCESS VARIABLE コマンドは、process引数に渡した番号のソースプロセスからsrcVar (srvVar2等) プロセス変数を読み込み、その現在の値をカレントプロセスのdstVar (dstVar2等) 変数に返します。

それぞれのソース変数は変数、配列、配列要素のいずれかを指定できます。ただし、この節で後述する制限事項を参照してください。

srcVar;dstVar変数の組み合わせにおいて、2つの変数は互換性のあるタイプである必要があり、互換性がない場合には、値を取得しても意味がなくなります。

カレントプロセスはソースプロセスの変数を"のぞき見"しています。ソースプロセスは別のプロセスが自分の変数のインスタンスを読み込んでいることについては何も警告されません。

4D Server: 4D Clientを使用し、サーバマシン上で実行される目的のプロセス (ストアドプロシージャ) の変数を読み込むことができます。このためには、process引数に渡すプロセス番号の前にマイナス記号を付けてください。
GET PROCESS VARIABLE、SET PROCESS VARIABLE、VARIABLE TO VARIABLEコマンドにより提供されるマシン間プロセス通信はクライアントからサーバへのみ行うことができます。ストアドプロシジャの読み書きを行うのは常にクライアントプロセスです。

Tip: サーバのプロセス番号がわからない場合でも、サーバのインタープロセス変数を使用することができます。このためには、processに任意の負の値を指定します。つまり、プロセス番号がわからなくてもGET PROCESS VARIABLEコマンドを使用してサーバのインタープロセス変数値を得ることができるということです。このことは、On Server Startupデータベースメソッドを使用して、ストアドプロシージャが起動されている場合に便利です。クライアントマシンではそのプロセスの番号が自動的にわからないため、process引数に任意の負の値を渡すことができます。

制限事項

GET PROCESS VARIABLEは、ソース変数としてローカル変数を受け付けません。

一方で受け取り側の変数はインタープロセス、プロセス、またはローカル変数を使用できます。値は変数にのみ受け取ることができ、フィールドで受け取ることはできません。

GET PROCESS VARIABLEは、任意のタイプのソースプロセス変数またはインタープロセス変数を受け付けますが、以下のタイプを除きます:

- ポインタ
- ポインタの配列
- 2次元配列

ソースプロセスは、ユーザプロセスである必要があります。カーネルプロセスはソースプロセスにはなれません。ソースプロセスが存在しない場合には、このコマンドは何も行いません。

Note: インタープリタモードでは、ソース変数が存在しない場合には未定義値が返されます。これはTypeを使って対応するソース変数をテストし、検出することができます。

例題 1

以下のコードは、プロセス番号が$vlProcessであるプロセスのテキスト変数vtCurStatusの値を読み込み、その値をカレントプロセスのプロセス変数vtInfoに返します:

```
GET PROCESS VARIABLE($vlProcess;vtCurStatus;vtInfo)
```

例題 2

以下のコードは上記の例と同じことをしますが、カレントプロセスで実行しているメソッドのローカル変数$vtInfoに値を返します:

```
GET PROCESS VARIABLE($vlProcess;vtCurStatus;$vtInfo)
```

例題 3

以下のコードは上記の例と同じことをしますが、カレントプロセスのvtCurStatus変数に値を返します:

```
GET PROCESS VARIABLE($vlProcess;vtCurStatus;vtCurStatus)
```

Note: 最初のvtCurStatusはソースプロセスにある変数のインスタンスを示しています。2番目のvtCurStatusはカレントプロセスにある変数のインスタンスを示しています。

例題 4

以下の例は、$vlProcessで示されるプロセスからプロセス配列の要素を順次読み込みます:

```
GET PROCESS VARIABLE($vlProcess;vl_IPCom_Array;$vlSize)
For($vlElem;1;$vlSize)
   GET PROCESS VARIABLE($vlProcess;at_IPCom_Array{$vlElem};$vtElem)
  ` Do something with $vtElem
End for
```

Note: この例では、プロセス変数vl_IPCom_Arrayには配列at_IPCom_Arrayのサイズが格納され、送信元プロセスによって管理されている必要があります。

例題 5

以下の例は上記の例と同じことをしますが、配列の要素を順番に読み込む代わりに配列を全体として読み込みます:

```
GET PROCESS VARIABLE($vlProcess;at_IPCom_Array;$anArray)
For($vlElem;1;Size of array($anArray))
  ` Do something with $anArray{$vlElem}
End for
```

例題 6

以下の例は、変数v1,v2,v3のソースプロセスインスタンスを読み込み、それらの値をカレントプロセスの同じ変数のインスタンスに返します:

```
GET PROCESS VARIABLE($vlProcess;v1;v1;v2;v2;v3;v3)
```

例題 7

DRAG AND DROP PROPERTIESコマンドの例題参照

Semaphore

Semaphore (semaphore {; tickCount}) -> 戻り値

引数	型	説明
semaphore	文字	→ テストと設定を行うセマフォ
tickCount	倍長整数	→ 最大待ち時間
戻り値	ブール	↩ FALSE: セマフォの設定に成功した TRUE: 既にセマフォが存在する

説明

セマフォは、ワークステーション (各ユーザのコンピュータ) 間、または同一ワークステーション上のプロセス間で共有されるフラグです。セマフォは、単に存在したり存在しなかったりするだけです。各ユーザが実行しているメソッドでセマフォの存在を調べることができます。セマフォを作成する、またはその存在の有無を調べることにより、ワークステーション間でのメソッドの通信が可能になります。

Semaphore 関数はすでにセマフォが存在する場合TRUEを返し、何も行いません。セマフォが存在しない場合、**Semaphore** はセマフォを作成しFALSEを返します。同時に1人のユーザしかセマフォを作成することはできません。**Semaphore**がFALSEを返すということは、セマフォが存在しなかったことを意味すると同時に、コマンド呼び出したプロセスに対して新たにセマフォ設定されたことを意味します。

Semaphoreは、セマフォが設定されていなければFALSEを返します。またコマンドを呼び出したプロセスが既にそのセマフォを設定している場合もFALSEを返します。semaphoreは先頭の$を含めて255文字以内に制限されています。これより長い文字列を指定すると、切り捨てられた文字列を使ってセマフォがテストされます。

オプションの引数tickCountは、semaphoreが既にセットされている時の待ち時間 (tick) を設定します。
この場合、関数はセマフォが解放されるか、またはTRUEを返す前に待ち時間が終了まで待ちます。

4Dには2種類のセマフォ、ローカルセマフォとグローバルセマフォがあります。

- ローカルセマフォは、同じワークステーション上のすべてのプロセスからアクセスすることができます (同一ワークステーション上に限られます) 。ローカルセマフォは、セマフォ名の先頭にドル記号 ($) を付けて作成します。ローカルセマフォは、同一ワークステーション上で実行しているプロセス間で処理を監視する際に使用します。例えばローカルセマフォを使用して、シングルユーザデータベースやワークステーション上のすべてのプロセスで共用するインタープロセス配列へのアクセスを監視します。
- グローバルセマフォは、すべてのユーザとそのプロセスからアクセスすることができます。グローバルセマフォはマルチユーザデータベースのユーザ間で処理を監視するために用います。

グローバルセマフォとローカルセマフォは理論的には同じものです。違いはその有効範囲にあります。

クライアント/サーバーモードでは、グローバルセマフォはすべてのクライアントおよびサーバーで実行しているすべてのプロセス間で共用されます。ローカルセマフォは、それが作成されたマシン上で実行しているプロセス間でのみ共用されます。

スタンドアロンモードの4Dでは、ユーザは一人だけなため、グローバルセマフォもローカルセマフォもその有効範囲は同じです。ただし、シングルとマルチの両方の形でデータベースを使用する場合は、用途によってグローバルセマフォとローカルセマフォを使い分けてください。

注: インターフェースやインタープロセス変数など、クライアントアプリケーションのローカルな状態を管理するためにセマフォを使用する場合、ローカルセマフォを利用することをお勧めします。このようなケースでグローバルセマフォを使用すると、不必要なネットーワークアクセスが行われるだけでなく、不必要に他のクライアントに影響を与えてしまいます。ローカルセマフォを使用すればこのような望ましくない副作用を避けることができます。

セマフォはレコードのアクセスの保護目的には使用しません。これは4Dと4D Serverが自動的に行います。セマフォは、複数のユーザが同じ処理を同時に実行するのを防ぐために用います。

例題 1

以下の例では、2人のユーザがProducts テーブルの価格を更新するのを防ぎます。以下のメソッドではセマフォを用いて、これを実現しています:

```
If(Semaphore("UpdatePrices")) ` セマフォの作成を試行
   ALERT("Another user is already updating prices. Retry later.")
Else
   DoUpdatePrices ` 料金の更新
   CLEAR SEMAPHORE("UpdatePrices")) ` セマフォをクリア
End if
```

例題 2

以下の例はローカルセマフォを使用します。複数のプロセスを持つデータベースで、To Doリストを管理する必要があるとします。このリストはテーブルではなく、インタープロセス配列で管理します。セマフォを使って同時にアクセスされるのを防ぎます。このような場合に、To Doリストは自分だけのものなため、ローカルセマフォで十分です。

インタープロセス配列はOn Startup データベースメソッドで初期化します:

```
ARRAY TEXT(<>ToDoList;0) ` The To Do list is initially empty
```

To Doリストに項目を追加するメソッドを次に示します:

```
 ` ADD TO DO LIST project method
 ` ADD TO DO LIST ( Text )
 ` ADD TO DO LIST ( To do list item )
C_TEXT($1)
If(Not(Semaphore("$AccessToDoList";300)))
 ` Wait 5 seconds if the semaphore already exists
   $vlElem:=Size of array(<>ToDoList)+1
   INSERT IN ARAY(<>ToDoList;$vlElem)
   <>ToDoList{$vlElem}:=$1
   CLEAR SEMAPHORE("$AccessToDoList") ` Clear the semaphore
End if
```

どのプロセスからも上記メソッドを呼び出せます。

例題 3

以下のメソッドは、セマフォーが存在する場合コードを実行せず、呼び出し元メソッドにエラーコードとテキストメッセージを返します。

シンタックス:

```
$L_Error:=Semaphore_proof(->$T_Text_error)

 // セマフォーを使用した保護構造
C_LONGINT($0;$L_MyError)
C_POINTER($1) // エラーメッセージ</p><p>C_TEXT($T_Sema_local;$T_Message)

 // メソッド開始
$L_MyError:=0
$T_Sema_local:="$tictac"

If(Semaphore($T_Sema_local;300))
 // 300 tickの待ち時間の間に、同じセマフォを作成したプロセスがWe expected 300 ticks but the semaphore
 // そのセマフォを解放しなかった場合、ここで停止する
   $L_MyError:=-1

Else

 // このメソッドは同時に複数プロセスで実行されることがない

 // このプロセスでセマフォーを設定したので、
 // このプロセスで必ずセマフォーを解放しなければならない

 // 処理を行う
   ...
 // セマフォーを解放する
   CLEAR SEMAPHORE($T_Sema_local)
End if

If($L_MyError=-1)
   $T_Message:="セマフォー "+$T_Sema_local+" が既に設定されていたためコードは実行されませんでした。"
Else
   $T_Message:="OK"
End if

$0:=$L_MyError
$1->:=$T_Message  // 呼び出し元メソッドはエラーコードとメッセージテキストを受け取る
```

SET PROCESS VARIABLE

SET PROCESS VARIABLE (process ; dstVar ; expr {; dstVar2 ; expr2 ; ... ; dstVarN ; exprN})

引数	型		説明
process	倍長整数	→	送り先のプロセス番号
dstVar	変数	→	送り先の変数
expr	変数	→	ソース式 (ソース変数)

説明

SET PROCESS VARIABLE コマンドは、引数expr1 (expr2等)に渡す値を、processに渡す番号の送り先プロセスのdstVar (dstVar2等) プロセス変数に書き込みます。

それぞれの送り先変数は変数、または配列要素のいずれかを指定できます。ただし、この節で後述する制限事項を参照してください。

srcVar;dstVarの組み合わせにおいて、式は送り先変数と互換性のあるタイプである必要があり、互換性がない場合には、意味のない値が設定されます。インタプリタモードでは、送り先変数が存在しない場合、変数が作成され式の値が設定されます。

カレントプロセスは送り先プロセスの変数を"のぞき見"しています。送り先プロセスは別のプロセスが自分の変数のインスタンスに書き込んでいることについては何も警告されません。

4D Server: 4D Clientを使用し、サーバマシン上で実行される目的のプロセス (ストアドプロシージャ) の変数に書き込むことができます。このためには、process引数に渡すプロセス番号の前にマイナス記号を付けてください。
GET PROCESS VARIABLE、SET PROCESS VARIABLE、VARIABLE TO VARIABLEコマンドにより提供されるマシン間プロセス通信はクライアントからサーバへのみ行うことができます。ストアドプロシジャの読み書きを行うのは常にクライアントプロセスです。

Tip: サーバのプロセス番号がわからない場合でも、サーバのインタープロセス変数を使用することができます。このためには、processに任意の負の値を指定します。つまり、プロセス番号がわからなくてもGET PROCESS VARIABLEコマンドを使用してサーバのインタープロセス変数値を処理することができるということです。このことは、On Server Startupデータベースメソッドを使用して、ストアドプロシージャが起動されている場合に便利です。クライアントマシンではそのプロセスの番号が自動的にわからないため、process引数に任意の負の値を渡すことができます。

制限事項

SET PROCESS VARIABLEは、送り先変数としてローカル変数を受け付けません。

SET PROCESS VARIABLEは、任意のタイプの送り先プロセスまたはインタープロセス変数を受け付けますが、以下のタイプは除きます:

- ポインタ
- すべての配列：あるプロセスから別のプロセスに配列を全体として書き込むには、VARIABLE TO VARIABLEコマンドを使用します。ただし、SET PROCESS VARIABLEコマンドは配列の要素を書き込むことはできます。
- ポインタ配列の要素または2次元配列の要素を書き込むことはできません。

送り先プロセスはユーザプロセスである必要があります。カーネルプロセスは送り先プロセスにはなれません。送り先プロセスが存在しない場合、エラーが生成されます。ON ERR CALLでインストールされたエラー処理メソッドを使用すると、このエラーをとらえることができます。

例題 1

下のコードは、番号が$vlProcessであるプロセスのテキスト変数vtCurStatusを(空の文字列に)設定します:

```
 SET PROCESS VARIABLE($vlProcess;vtCurStatus;"")
```

例題 2

以下のコードは、番号が$vlProcessであるプロセスのテキスト変数vtCurStatusを、カレントプロセスで実行中のメソッドの変数$vtInfoの値に設定します:

```
 SET PROCESS VARIABLE($vlProcess;vtCurStatus;$vtInfo)
```

例題 3

以下のコードは、番号が$vlProcessであるプロセスのテキスト変数vtCurStatusをカレントプロセスの同じ変数の値に設定します:

```
 SET PROCESS VARIABLE($vlProcess;vtCurStatus;vtCurStatus)
```

Note: 最初のvtCurStatusは送り先プロセスにある変数のインスタンスを示しています。2番目のvtCurStatusはカレントプロセスにある変数のインスタンスを示しています。

例題 4

以下の例は$vlProcessで示されるプロセスのプロセス配列の要素を順次大文字に設定
します:

```
 GET PROCESS VARIABLE($vlProcess;vl_IPCom_Array;$vlSize)
 For($vlElem;1;$vlSize)
    GET PROCESS VARIABLE($vlProcess;at_IPCom_Array{$vlElem};$vtElem)
    SET PROCESS VARIABLE($vlProcess;at_IPCom_Array{$vlElem};Uppercase($vtElem))
 End for
```

Note: この例では、プロセス変数vl_IPCom_Arrayには配列at_IPCom_Arrayのサイズが格納され、ソース/送信先プロセスによって管理されている必要があります。

例題 5

以下の例は、現在のプロセスの同じ変数のインスタンスを使用して、送り先プロセスの変数v1、v2、v3のインスタンスに書き込みます:

```
 SET PROCESS VARIABLE($vlProcess;v1;v1;v2;v2;v3;v3)
```

Test semaphore

Test semaphore (semaphore) -> 戻り値

引数	型		説明
semaphore	文字	→	テストするセマフォ
戻り値	ブール	↩	True: セマフォが存在する, False: セマフォは存在しない

説明

Test semaphore コマンドは、セマフォの存在をテストします。

Semaphore関数とTest semaphore関数の違いは、Test semaphoreはセマフォが存在しない場合にはsemaphoreを作成しないということです。semaphoreが存在している場合、関数はTrueを返します。そうでない場合はFalseを返します。

例題

下記の例は、セマフォを変更せずにプロセスの状態 (この場合は、コードを変更している最中かどうか) を知ることを可能にするものです:

```
$Win:=Open window(x1;x2;y1;y2;-Palette window)
Repeat
   If(Test semaphore("Encrypting code"))
      POSITION MESSAGE($x3;$y3)
      MESSAGE("Encrypting code being modified.")
   Else
      POSITION MESSAGE($x3;$y3)
      MESSAGE("Modification of the encrypting code authorized.")
   End if
Until(StopInfo)
CLOSE WINDOW
```

VARIABLE TO VARIABLE

VARIABLE TO VARIABLE (process ; dstVar ; srcVar {; dstVar2 ; srcVar2 ; ... ; dstVarN ; srcVarN})

引数	型		説明
process	倍長整数	⇨	送り先プロセス番号
dstVar	変数	⇨	送り先変数
srcVar	変数	⇨	ソース変数

説明

VARIABLE TO VARIABLEコマンドは、引数srcVar1 srcVar2に渡す値を、processに渡す番号を持つ送り先プロセスのdstVar (dstVar2等) プロセス変数に書き込みます。

VARIABLE TO VARIABLEは、SET PROCESS VARIABLEコマンドと同じ動作をしますが、以下の点が異なります:

- SET PROCESS VARIABLEコマンドは引数にソース式を渡すため、配列全体を渡すことができません。これに対して、VARIABLE TO VARIABLEコマンドは明示的に引数としてソース変数を渡すため、配列を全体として渡すことができます。
- SET PROCESS VARIABLEコマンドの各送り先変数は変数または配列要素を指定することができますが、配列全体は指定できません。VARIABLE TO VARIABLEコマンドの各送り先変数は変数、配列または配列要素を指定することができます。

カレントプロセスは送り先プロセスの変数を"のぞき見"しています。送り先プロセスは別のプロセスが自分の変数のインスタンスに書き込んでいることについては何も警告されません。

4D Server: GET PROCESS VARIABLE、SET PROCESS VARIABLE、VARIABLE TO VARIABLEコマンドにより提供されるマシン間プロセス通信はクライアントからサーバへのみ行うことができます。ストアドプロシジャの読み書きを行うのは常にクライアントプロセスです。

srcVar;dstVarの組み合わせにおいて、ソース変数は送り先変数と互換性のあるタイプである必要があり、互換性がない場合には、意味のない値が設定されます。
インタプリタモードでは、送り先変数が存在しない場合、変数が作成されソース変数の値が設定されます。

カレントプロセスは送り先プロセスの変数を"のぞき見"しています。送り先プロセスは別のプロセスが自分の変数のインスタンスに書き込んでいることについては何も警告されません。

制限事項

VARIABLE TO VARIABLE は、送り先変数としてローカル変数を受け付けません。

VARIABLE TO VARIABLE は、任意のタイプの送り先プロセスまたはインタープロセス変数を受け付けますが、以下のタイプは除きます:

- ポインタ
- ポインタ配列
- 2次元配列

送り先プロセスは、ユーザプロセスである必要があります。カーネルプロセスは、送り先プロセスにはなれません。送り先プロセスが存在しない場合には、エラーが生成されます。ON ERR CALLコマンドでインストールされたエラー処理メソッドを使用すると、このエラーをとらえることができます。

例題

以下の例は、ローカル変数$vlProcessで示されたプロセスからプロセス配列を読み込み、配列要素を順番に大文字に変換して、配列を全体として書き込みます:

```
GET PROCESS VARIABLE($vlProcess;at_IPCom_Array;$anArray)
For($vlElem;1;Size of array($anArray))
   $anArray{$vlElem}:=Uppercase($anArray{$vlElem})
End for
VARIABLE TO VARIABLE($vlProcess;at_IPCom_Array;$anArray)
```

プロセス (ユーザインタフェース)

- BRING TO FRONT
- Frontmost process
- HIDE PROCESS
- SHOW PROCESS

BRING TO FRONT

BRING TO FRONT (process)

引数	型		説明
process	倍長整数	→	最前面に移動させるプロセスのプロセス番号

説明

BRING TO FRONTはprocessに属するすべてのウィンドウを最前面に配置します。ウィンドウの順序は変わりません。このプロセスが既に最前のプロセスの場合は、このコマンドは何も行いません。プロセスが非表示の場合に、SHOW PROCESSコマンドでプロセスを表示しないとBRING TO FRONTコマンドは効果がありません。

このコマンドを使用して、アプリケーションプロセスとデザインプロセスを最前面にすることができます。

例題

次の例は、メニューから実行できるプロジェクトメソッドです。これは、最前のプロセスが<>vlAddCust_PIDプロセスかどうかを調べています。そうでなければ、それを前面に配置します:

```
If(Frontmost process#<>vlAddCust_PID)
   BRING TO FRONT(<>vlAddCust_PID)
End if
```

Frontmost process

Frontmost process {(*)} -> 戻り値

引数	型		説明
*	演算子	→	フローティングウインドウ以外の最前面のプロセス番号
戻り値	整数	↩	最前面にあるウィンドウのプロセス番号

説明

Frontmost processは、ウィンドウが最前面にあるプロセスの番号を返します。

1つ以上のフローティングウィンドウがある場合は、次の2種類のウィンドウレイヤがあります:

- 通常のウィンドウ
- フローティングウィンドウ

フローティングウィンドウのフォームメソッドやオブジェクトメソッドからFrontmost processコマンドを使用すると、このコマンドはフローティングウィンドウレイヤ内の最前面のフローティングウィンドウのプロセス番号を返します。オプションの * 引数を指定すると、この関数は、通常ウィンドウレイヤ内の最前面のアクティブウィンドウのプロセス番号を返します。

例題

BRING TO FRONTの例題参照

HIDE PROCESS

HIDE PROCESS (process)

引数	型		説明
process	倍長整数	→	隠すプロセスのプロセス番号

説明

HIDE PROCESSはprocessに属するすべてのウィンドウを非表示にします。processのすべてのインタフェース要素は、次にSHOW PROCESSコマンドを実行するまで非表示となります。そのプロセスのメニューバーも非表示になります。したがって、プロセスが非表示になっているときにウィンドウを開いても画面が再描画されたり表示されません。プロセスが既に非表示になっている場合、このコマンドは何も実行しません。

ただし、デバッグウィンドウだけは例外です。processを非表示にしてもデバッグウィンドウが表示されると、processは表示され最前面のプロセスとなります。

processを作成した時点でそれを全く表示したくなければ、HIDE PROCESSコマンドをプロセスメソッドの最初のコマンドにします。このコマンドはメインプロセスおよびキャッシュマネージャプロセスを非表示にすることはできません。

プロセスを非表示にした場合でも、そのプロセスは実行し続けます。

例題

次の例は、カレントプロセスのすべてのウインドウを非表示にします:

```
HIDE PROCESS(Current process)
```

SHOW PROCESS

SHOW PROCESS (process)

引数	型	説明
process	倍長整数	→ 表示させるプロセスのプロセス番号

説明

SHOW PROCESSはprocessに属する全ウィンドウを表示します。このコマンドはprocessのウィンドウを最前面ウィンドウにするわけではありません。これを行うにはBRING TO FRONTコマンドを使用します。
プロセスが既に表示されている場合は、このコマンドは何も実行しません。

例題

次の例は、以前に非表示になっていれば、Customersプロセスを表示します。Customersプロセスのプロセス参照は事前にプロセス変数<>*Customers*に格納されています:

```
SHOW PROCESS(<>Customers)
```

ペーストボード

ペーストボードの管理
APPEND DATA TO PASTEBOARD
CLEAR PASTEBOARD
Get file from pasteboard
GET PASTEBOARD DATA
GET PASTEBOARD DATA TYPE
GET PICTURE FROM PASTEBOARD
Get text from pasteboard
Pasteboard data size
SET FILE TO PASTEBOARD
SET PICTURE TO PASTEBOARD
SET TEXT TO PASTEBOARD

ベーストボードの管理

"ベーストボード" テーマのコマンドは、コピー/ベーストアクション (クリップボード管理) とアプリケーション間のドラッグ＆ドロップの管理両方に使用できます。

4Dは2つのデータベーストボードを使用します。1つはコピーあるいはカットされたデータ用で、これは以前のバージョンからある実際のクリップボードです。もう1つはドラッグされドロップされたデータ用です。
これら2つのベーストボードは同じコマンドを使用して管理されます。コンテキストにより、どちらかにアクセスします:

- ドラッグ＆ドロップベーストボードには、On Begin Drag Over, On Drag over, On Dropフォームイベントや内でのみアクセスできます。これらのコンテキスト外では、ドラッグ＆ドロップベーストボードは使用できません。
- コピー/ベーストベーストボードには、他のすべてのケースでアクセスできます。ドラッグ＆ドロップベーストボードと異なり、そこに置かれたデータはクリアされるか再利用されるまで、セッション中保持されます。

データのタイプ

ドラッグ＆ドロップアクション中、異なるタイプのデータがベーストボードに置かれたり、あるいはベーストボードから読み込まれます。データタイプには複数の方法でアクセスします:

- 4Dシグネチャ: 4Dシグネチャは4Dアプリケーションにより参照されるデータタイプを示す文字列です。4DシグネチャはMac OSおよびWindowsで同じであるため、マルチプラットフォームアプリケーションの開発に適しています。4Dシグネチャはこの節の最後に示します。
- UTI (Uniform Type Identifier, Mac OSのみ): Apple社が定めるUTI標準は、ネイティブタイプのオブジェクトごとに文字列を割り当てたものです。例えばGIF ピクチャにはUTI タイプ "com.apple.gif"が割り当てられています。UTIはApple社のドキュメント、あるいは関連するエディタで公開されています。
- 数値またはフォーマット名 (Windowsのみ): Windowsでは、ネイティブデータタイプは数値 ("3", "12", 等) と名前 ("Rich Text Edit")で参照されます。デフォルトでMicrosoft社は標準データフォーマットと呼ばれるネイティブタイプを複数定義しています。さらにサードパーティーエディタはシステムにフォーマットを "保存" し、対応する番号を得ることもできます。この点に関する詳細とネイティブタイプについては、Microsoft developer documentation (特に http://msdn2.microsoft.com/en-us/library/ms649013.aspx) を参照してください。

Note: 4Dコマンド中では、Windowsのフォーマット番号はテキストとして処理されます。

"ベーストボード" テーマのすべてのコマンドは、これらのデータタイプのそれぞれ1つを使用できます。GET PASTEBOARD DATA TYPE コマンドを使用して、ベーストボードに格納されているデータのデータタイプを取得できます。

Note: 4文字のタイプ (TEXT, PICT やカスタムタイプ) は以前のバージョンの4Dとの互換性のために保持されています。

4D シグネチャ

以下は標準の4D シグネチャとその説明です:

シグネチャ	説明
"com.4d.private.text.native"	ネイティブ文字セットのテキスト
"com.4d.private.text.utf16"	Unicode文字セットのテキスト
"com.4d.private.text.rtf"	リッチテキスト
"com.4d.private.picture.pict"	PICT ピクチャフォーマット
"com.4d.private.picture.png"	PNG ピクチャフォーマット
"com.4d.private.picture.gif"	GIF ピクチャフォーマット
"com.4d.private.picture.jfif"	JPEG ピクチャフォーマット
"com.4d.private.picture.emf"	EMF ピクチャフォーマット
"com.4d.private.picture.bitmap"	BITMAP ピクチャフォーマット
"com.4d.private.picture.tiff"	TIFF ピクチャフォーマット
"com.4d.private.picture.pdf"	PDF ドキュメント
"com.4d.private.file.url"	ファイルパスメ名

APPEND DATA TO PASTEBOARD

APPEND DATA TO PASTEBOARD (dataType ; data)

引数	型		説明
dataType	文字	⇨	追加するデータのタイプ
data	BLOB	⇨	ペーストボードに追加するデータ

説明

APPEND DATA TO PASTEBOARD コマンドは、dataTypeで指定されたデータタイプでdataBLOB内にあるデータをペーストボードに追加します。

Note: コピー/ペースト操作の場合、ペーストボードはクリップボードと同じです。

dataTypeには、追加するデータのタイプを指定する値を渡します。4D シグネチャ, UTI タイプ (Mac OS), フォーマット名/番号 (Windows), 4文字のタイプ (互換性) を渡すことができます。これらのデータタイプについてはの節を参照してください。

Windowsユーザへの注意: コマンドをテキストタイプのデータに対して使用するとき (dataType が"TEXT"、com.4d.private.text.native または com.4d.private.text.utf16)、Blob引数dataに含まれる文字列はNULL文字で終了していなければなりません。

通常、同一データの複数のインスタンスをペーストボードに追加、またはテキストやピクチャ以外のタイプのデータを追加するときには、APPEND DATA TO PASTEBOARDコマンドを使用します。ペーストボードに新しいデータを追加するには、まず最初にCLEAR PASTEBOARDコマンドを使用してペーストボードを消去する必要があります。

消去と追加を実行するには:

- テキスト: SET TEXT TO PASTEBOARDコマンドを使用します。
- ピクチャ: SET PICTURE TO PASTEBOARDコマンドを使用します。
- ファイルパス名 (ドラッグ＆ドロップ) : SET FILE TO PASTEBOARDコマンドを使用します。

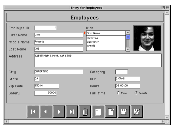
しかし、BLOBにテキストやピクチャが含まれている場合、APPEND DATA TO PASTEBOARD コマンドを使用してテキストやピクチャをペーストボードに追加できることに留意してください。

例題

ペーストボードコマンドとBLOBを使用すると、洗練されたカット/コピー/ペーストの仕組みを構築でき、たった1つのデータではなく構造化されたデータを扱うことができます。以下の例では、2つのプロジェクトメソッドSET RECORD TO PASTEBOARDとGET RECORD FROM PASTEBOARDは、ペーストボードとの間でコピーするためにレコード全体を1つのデータとして扱うことができます。

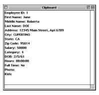
```
  ` SET RECORD TO <span class="rte4d_cmd">PASTEBOARD</span><gen9> プロジェクトメソッド
  ` SET RECORD TO </gen9><span class="rte4d_cmd">PASTEBOARD</span><gen9> ( 数値 )
  ` SET RECORD TO </gen9><span class="rte4d_cmd">PASTEBOARD</span><gen9> ( テーブル 数値 )

 C_LONGINT($1;$vlField;$vlFieldType)
 C_POINTER($vpTable;$vpField)
 C_STRING(255;$vsDocName)
 C_TEXT($vtRecordData;$vtFieldData)
 C_BLOB($vxRecordData)

  ` ペーストボードをクリア (カレントレコードがない場合には空のままとなる)
 CLEAR PASTEBOARD
  ` 引数で渡されたテーブル番号のテーブルポインタを得る
 $vpTable:=Table($1)
  ` テーブルのカレントレコードがあれば
 If((Record number($vpTable->)>=0)|(Is new record($vpTable->)))
  ` レコードのテキストイメージを保持するテキスト変数を初期化
    $vtRecordData:=""
  ` レコードのフィールドごとに
    For($vlField;1;Get last field number($1))
  ` フィールドの型を取得
       GET FIELD PROPERTIES($1;$vlField;$vlFieldType)
  ` フィールドのポインタを取得
       $vpField:=Field($1;$vlField)
  ` フィールド型に基づき、適切な方法でデータをコピー
       Case of
          :(($vlFieldType=Is Alpha Field)|($vlFieldType=Is Text))
             $vtFieldData:=$vpField->
          :(($vlFieldType=Is Real)|($vlFieldType=Is Integer)|($vlFieldType=Is LongInt)
             |($vlFieldType=Is Date)|($vlFieldType=Is Time))
             $vtFieldData:=String($vpField->)
          :($vlFieldType=Is Boolean)
             $vtFieldData:=String(Num($vpField->);"Yes;;No")
          Else
  ` 他のデータタイプは無視
             $vtFieldData:=""
       End case
  ` レコードのテキストイメージを保持するテキスト変数にフィールドデータを追加
       $vtRecordData:=$vtRecordData+Field name($1;$vlField)+":"+Char(9)+$vtFieldData+CR
  ` Note: CRメソッドは、Mac OS上ではChar(13)を、Windows上ではChar(13)+Char(10)を返す
    End for
  ` テキストイメージをペーストボードに置く
    SET TEXT TO PASTEBOARD($vtRecordData)
  ` Temporary フォルダのスクラップファイル名
    $vsDocName:=Temporary folder+"Scrap"+String(1+(Random%99))
  ` スクラップファイルがあれば削除する (ここでエラーをテストすべき)
    DELETE DOCUMENT($vsDocName)
  ` スクラップファイルを作成
    SET CHANNEL(10;$vsDocName)
  ` スクラップファイルにレコード全体を送信
    SEND RECORD($vpTable->)
  ` スクラップファイルを閉じる
    SET CHANNEL(11)
  ` スクラップファイルをBLOB読み込む
    DOCUMENT TO BLOB($vsDocName;$vxRecordData)
  ` スクラップファイルはもう必要ない
    DELETE DOCUMENT($vsDocName)
  ` ペーストボードにレコードの完全なイメージを追加
  ` Note: ここではデータ型に"4Drc" を使用しています
    APPEND DATA TO PASTEBOARD("4Drc";$vxRecordData)
  ` この時点でペーストボードには以下が含まれます:
  ` (1) レコードのテキストイメージ (以下のスクリーンショットで見られるような)
  ` (2) レコード全体のイメージ (ピクチャや BLOB フィールドを含む)
 End if</gen9>
```

以下のようなレコードを表示させた時:



Employee ID 1
First Name Jane
Middle Name Roberta
Last Name DOE
Address 12345 Main Street, Apt 6789
Kids
First Name
Christina
Sylvester
Arnold
City CUPERTINO
State CA
Zip Code 95014
Salary 50000
Category
DOB 2/5/61
Hours 08:00:00
Full time ○ Male ● Female

SET RECORD TO PASTEBOARD メソッドを[Employees] テーブルに適用すると、ペーストボードには以下のようなレコードのテキストイメージとレコード全体のイメージが含まれます。



Employee ID: 1
First Name: Jane
Middle Name: Roberta
Last Name: DOE
Address: 12345 Main Street, Apt 6789
City: CUPERTINO
State: CA
Zip Code: 95014
Salary: 50000
Category: 4
DOB: 2/5/61
Hours: 08:00:00
Full Time: No
Photo:
Kids:

GET RECORD FROM PASTEBOARDメソッドを使用して、このレコードイメージを他のレコードにペーストできます:

```
 <gen9> ` GET RECORD FROM </gen9><span class="rte4d_cmd">PASTEBOARD</span><gen9>メソッド
  ` GET RECORD FROM </gen9><span class="rte4d_cmd">PASTEBOARD</span><gen9>( 数値 )
  ` GET RECORD FROM </gen9><span class="rte4d_cmd">PASTEBOARD</span><gen9>( テーブル 数値 )
 C_LONGINT($1;$vlField;$vlFieldType;$vlPosCR;$vlPosColon)
 C_POINTER($vpTable;$vpField)
 C_STRING(255;$vsDocName)
 C_BLOB($vxPasteboardData)
 C_TEXT($vtPasteboardData;$vtFieldData)

  ` 引数として渡されたテーブル番号のテーブルポインタを得る
 $vpTable:=Table($1)
  ` カレントレコードがあれば
 If((Record number($vpTable->)>=0)|(Is new record($vpTable->)))
    Case of
  ` ペーストボードに完全なレコードイメージが含まれているか?
       :(Pasteboard data size("4Drc")>0)
  ` 含まれていればペーストボードの中身を取り出す
          GET PASTEBOARD DATA("4Drc";$vxPasteboardData)
  ` Temporary フォルダ内のスクラップファイル名
          $vsDocName:=Temporary folder+"Scrap"+String(1+(Random%99))
  ` スクラップファイルが存在すれば削除する (ここでエラーをテストすべき)
          DELETE DOCUMENT($vsDocName)
  ` スクラップファイルにBLOBを保存
          BLOB TO DOCUMENT($vsDocName;$vxPasteboardData)
  ` スクラップファイルを開く
          SET CHANNEL(10;$vsDocName)
  ` スクラップファイルからレコード全体を取り込む
          RECEIVE RECORD($vpTable->)
  ` スクラップファイルを閉じる
          SET CHANNEL(11)
  ` スクラップファイルはもう必要ない
          DELETE DOCUMENT($vsDocName)
  ` ペーストボードにTEXTが含まれているか?
       :(Pasteboard data size("TEXT")>0)
  ` ペーストボードからテキストを取り出す
          $vtPasteboardData:=Get text from pasteboard
  ` インクリメントするフィールド番号を初期化
          $vlField:=0
          Repeat
  ` テキスト中で次のフィールド行を探す
             $vlPosCR:=Position(CR;$vtPasteboardData)
             If($vlPosCR>0)
  ` フィールド行を取り出す
                $vtFieldData:=Substring($vtPasteboardData;1;$vlPosCR-1)
  ` コロン ":" があれば
                $vlPosColon:=Position(":";$vtFieldData)
                If($vlPosColon>0)
  ` フィールドデータのみを取り出す (フィールド名を削除)
                   $vtFieldData:=Substring($vtFieldData;$vlPosColon+2)
                End if
  ` フィールド番号をインクリメント
                $vlField:=$vlField+1
  ` ペーストボードには必要以上のデータが含まれていることがある...
                If($vlField<=Get last field number($vpTable))
  ` フィールドタイプを取得
                   GET FIELD PROPERTIES($1;$vlField;$vlFieldType)
  ` フィールドへのポインタを取得
                   $vpField:=Field($1;$vlField)
  ` フィールドのデータ型に基づき、適切な方法でデータをコピー
                   Case of
                      :(($vlFieldType=Is Alpha Field)|($vlFieldType=Is Text))
                         $vpField->:=$vtFieldData
                      :(($vlFieldType=Is Real)|($vlFieldType=Is Integer)|($vlFieldType=Is LongInt))
                         $vpField->:=Num($vtFieldData)
                      :($vlFieldType=Is Date)
                         $vpField->:=Date($vtFieldData)
                      :($vlFieldType=Is Time)
                         $vpField->:=Time($vtFieldData)
                      :($vlFieldType=Is Boolean)
                         $vpField->:=($vtFieldData="Yes")
                      Else
  ` 他のフィールドタイプは無視
                   End case
                Else
  ` すべてのフィールドタイプに代入したのでループを抜ける
                   $vtPasteboardData:=""
                End if
  ` 取り出したテキストを削除
                $vtPasteboardData:=Substring($vtPasteboardData;$vlPosCR+Length(CR))
             Else
  ` 区切り文字が見つからないのでループを抜ける
                $vtPasteboardData:=""
             End if
  ` データがある間ループする
          Until(Length($vtPasteboardData)=0)
       Else
          ALERT("The pasteboard does not any data that can be pasted as a record.")
    End case
 End if</gen9>
```

システム変数およびセット

ペーストボードにBLOBデータが正しく追加されると、OKシステム変数は1に設定されます。そうでなければ0が設定され、エラーが生成されます。

エラー管理

ペーストボードにBLOBを追加するのに十分なメモリがない場合、エラー -108 が生成されます。

CLEAR PASTEBOARD

CLEAR PASTEBOARD
このコマンドは引数を必要としません

説明

CLEAR PASTEBOARD コマンドは、クリップボードの内容をすべて消去します。クリップボードに同じデータの複数のインスタンスが含まれる場合には、すべてのインスタンスが消去されます。CLEAR PASTEBOARDコマンドを呼び出した後、クリップボードは空になります。

APPEND DATA TO PASTEBOARDコマンドを使用して新しいデータをクリップボードに追加する前に、CLEAR PASTEBOARDコマンドを1回呼び出す必要があります。これはAPPEND DATA TO PASTEBOARDコマンドが新しいデータを追加する前にクリップボードを消去しないためです。

CLEAR PASTEBOARDを1回呼び出してから、APPEND DATA TO PASTEBOARDコマンドを複数回呼び出すと、異なるフォーマットで同じデータをカットまたはコピーすることができます。

一方SET TEXT TO PASTEBOARDとSET PICTURE TO PASTEBOARDコマンドは、クリップボードにデータを追加する前に自動的にクリップボードを消去します。

例題 1

以下の例はクリップボードを消去し、次にデータをペーストボードに追加します:

```
CLEAR PASTEBOARD ` ペーストボードを空にする
APPEND DATA TO PASTEBOARD("com.4d.private.picture.gif";$vxSomeData) ` gif ピクチャを追加
APPEND DATA TO PASTEBOARD("com.4d.private.text.rtf";$vxSylkData) ` RTFテキストを追加
```

例題 2

APPEND DATA TO PASTEBOARD コマンドの例題参照

Get file from pasteboard

Get file from pasteboard (xIndex) -> 戻り値

引数	型		説明
xIndex	倍長整数	→	ドラッグアクションに含まれるx番目のファイル
戻り値	文字	↩	ペーストボードから取り出した、ファイルのパス名

説明

Get file from pasteboard コマンドは、ドラッグ＆ドロップ処理に含まれるファイルの完全パス名を返します。複数のファイルを同時に選択し、移動することができます。xIndex 引数を使用して選択したファイル中でファイルを指定することができます。

ペーストボードにX番目のファイルがない場合、コマンドは空の文字列を返します。

例題

以下の例題は、ドラッグ＆ドロップ処理に含まれるすべてのファイルのパス名を配列に格納します:

```
ARRAY TEXT($filesArray;0)
C_TEXT($vfileArray)
C_INTEGER($n)
$n:=1
Repeat
   $vfileArray:=Get file from pasteboard($n)
   If($vfileArray#"")
      APPEND TO ARRAY($filesArray;$vfileArray)
      $n:=$n+1
   End if
Until($vfileArray="")
```

GET PASTEBOARD DATA

GET PASTEBOARD DATA (dataType ; data)

引数	型		説明
dataType	文字	→	ペーストボードから取り出すデータのタイプ
data	BLOB	←	ペーストボードから取り出されたデータ

説明

GET PASTEBOARD DATA コマンドはペーストボード中dataTypeで指定したタイプのデータを、BLOB フィールドまたは変数 data に返します。

dataTypeには取り出すデータのタイプを指定します。4D シグネチャ、UTI タイプ (Mac OS), フォーマット名/番号 (Windows), または4文字のタイプ(互換性)を指定できます。これらのタイプについてはの節を参照してください。

例題

以下の2つのオブジェクトメソッドはそれぞれフォーム上のasOptions 配列 (ポップアップメニューあるいはドロップダウンリスト) からデータをコピーあるいは配列へデータをペーストします:

```
  ` bCopyasOptions オブジェクトメソッド
 If(Size of array(asOptions)>0) ` コピーするものがあるか?
    VARIABLE TO BLOB(asOptions;$vxClipData) ` 配列要素をBLOBに格納
    CLEAR PASTEBOARD ` ペーストボードを空にする
    APPEND DATA TO PASTEBOARD("artx";asOptions) ` データ型は任意に選択されています
 End if

  ` bPasteasOptions オブジェクトメソッド
 If(Pasteboard data size("artx")>0) ` ペーストボードに"artx" タイプのデータがあるか?
    GET PASTEBOARD DATA("artx";$vxClipData) ` ペーストボードからデータを取り出す
    BLOB TO VARIABLE($vxClipData;asOptions) ` BLOBデータから配列を作成
    asOptions:=0 ` 配列の選択要素をリセット
 End if
```

システム変数およびセット

データが正しく取り出せるとOKシステム変数は1に設定されます。そうでなければ0が設定されエラーが生成されます。

エラー管理

データを取得するための十分なメモリがなければエラー -108 が生成されます。

GET PASTEBOARD DATA TYPE

GET PASTEBOARD DATA TYPE (4Dsignatures ; nativeTypes {; formatNames})

引数	型		説明
4Dsignatures	テキスト配列	⇐	データタイプの4D シグネチャ
nativeTypes	テキスト配列	⇐	ネイティブデータタイプ
formatNames	テキスト配列	⇐	フォーマット名 (Windowsのみ), Mac OSでは空の文字列

説明

GET PASTEBOARD DATA TYPE コマンドは、ペーストボードに含まれるデータタイプリストを取得するために使用します。このコマンドは一般的にドラッグ＆ドロップのコンテキストで、ドロップ先オブジェクトのOn Drop または On Drag Over フォームイベント内で使用されます。特に、ペーストボードに特定のデータタイプが存在するかどうかをチェックするために使用します。

このコマンドは2つまたは3つの配列に、複数の異なるフォーマットのデータタイプを返します:

- 4Dsignatures 配列には内部的な4D シグネチャ (例えば"com.4d.private.picture.gif") を使用して表現されたデータタイプが返されます。4Dが認識できないデータタイプの場合、空の文字列 ("") が配列に返されます。
- nativeTypes 配列にはネイティブタイプを使用して表現されたデータタイプが返されます。ネイティブタイプのフォーマットはMac OSとWindowsで異なります:
    - Mac OSでは、ネイティブタイプはUTI (Uniform Type Identifier) として表現されます。
    - Windowsでは、ネイティブタイプはフォーマット名に割り当てられた番号として表現されます。nativeTypes 配列にはこの番号が文字列 ("3", "12", 等) として格納されます。もし明確なラベルが必要な場合は、オプションのformatNames 配列を使用することをお勧めします。この配列にはWindowsにおけるネイティタイプのフォーマット名が格納されます。

  nativeTypes 配列では、4Dが参照できないタイプを含む、ペーストボード中のすべてのデータタイプ を知ることができます。
- Windowsでは、formatNames 配列を渡して、ペーストボード中のデータタイプ名を取得することができます。この配列に返される値は、例えばフォーマット選択ポップアップメニューを作成するために使用できます。Mac OSでは、formatNames 配列に空の文字列が返されます。

サポートされるデータタイプに関する詳細はの節を参照してください。

GET PICTURE FROM PASTEBOARD

GET PICTURE FROM PASTEBOARD (picture)

引数	型		説明
picture	ピクチャー	⇐	ペーストボードから取り出したピクチャ

説明

GET PICTURE FROM PASTEBOARD は、ペーストボード内に存在するピクチャーをpictureフィールドや変数に返します。

注: コピー/ペースト操作では、ペーストボードはクリップボードと同じです。

例題

以下のボタンオブジェクトメソッドは、ペーストボード中にピクチャ (jpeg または gif フォーマット)があれば、フィールド [Employees]Photoに代入します:

```
 If(Pasteboard data size("com.4d.private.picuture.jpeg")>0)|(Pasteboard data
size("com.4d.private.picture.gif")>0))
    GET PICTURE FROM PASTEBOARD([Employees]Photo)
 Else
    ALERT("The pasteboard does not contain any pictures.")
 End if
```

システム変数およびセット

ピクチャが正しく取り出されるとOKに1が、そうでなければ0が設定されます。

エラー管理

データを取り出すための十分なメモリがない場合には、エラー-108が生成されます。

Get text from pasteboard

Get text from pasteboard -> 戻り値

引数	型		説明
戻り値	文字	↩	ペーストボード中のテキスト (あれば)

説明

Get text from pasteboard は、ペーストボードに存在するテキストを返します。

Note: コピー/ペースト操作では、ペーストボードはクリップボードと同じです。

ペーストボードにRTFフォーマットなどのリッチテキストが含まれる場合、コピー先がサポートすれば、ドロップやペーストされたときテキストの属性は保持されます。

4Dのテキスト変数やフィールドは、2GBまでを含めることができます。

システム変数およびセット

テキストが正しく取り出されるとOKに1が、そうでなければ0が設定されます。

エラー管理

データを取り出すための十分なメモリがない場合には、エラー-108が生成されます。

Pasteboard data size

Pasteboard data size (dataType) -> 戻り値

引数	型		説明
dataType	文字	→	データタイプ
戻り値	倍長整数	↩	ペーストボード中のデータサイズ (バイト) またはエラーコード

説明

Pasteboard data size コマンドを使用して、dataType に渡したデータがペーストボード内に存在するかどうかを調べることができます。

Note: コピー/ペースト操作では、ペーストボードはクリップボードと同じです。

ペーストボードが空か指定したタイプのデータが含まれない場合、コマンドはエラー-102を返します。ペーストボードに指定したタイプのデータが含まれる場合、コマンドはバイト単位でデータのサイズを返します。

dataTypeにはチェックするデータのタイプを指定します。4D シグネチャ、UTI タイプ (Mac OS), フォーマット名/番号 (Windows), または4文字のタイプ(互換性)を指定できます。これらのタイプについてはペーストボードの管理を参照してください。

指定したタイプのデータがペーストボードに存在することを確認した後は、以下のいずれか1つのコマンドを使用し、そのデータをペーストボードから取り出すことができます:

- ペーストボードにあるデータがテキストタイプの場合には、テキスト値を返すGet text from pasteboardか、BLOBにテキストを返すGET PASTEBOARD DATAを使用してそのデータを取得できます。
- ペーストボードにあるデータがピクチャタイプの場合には、ピクチャをピクチャフィールドまたは変数に返すGET PICTURE FROM PASTEBOARDか、ピクチャをBLOBに返すGET PASTEBOARD DATAを使用してそのデータを取得できます。
- ペーストボードがファイルパス名を含む場合、ファイルパス名を返すGet file from pasteboard コマンドを使用します。
- 上記以外の任意のデータタイプに対しては、データをBLOBに返すGET PASTEBOARD DATAを使用します。

例題 1

以下のコードは、ペーストボードにJPEGピクチャが存在するかどうかをテストし、存在する場合にはそのピクチャを4D変数にコピーします:

```
If(Pasteboard data size("com.4d.private.picture.jfif")>0) ` ペーストボードにJPEGピクチャがあるか?
   GET PICTURE FROM PASTEBOARD($vPicVariable) ` もしあれば取り出す
Else
   ALERT("There is no picture in the pasteboard.")
End if
```

注: 汎用の'PICT'タイプ (またはPicture data定数) をこのコマンドに渡すと、ペーストボードにピクチャーが含まれる場合、コマンドは常にサイズではなく1を返します。

例題 2

通常、アプリケーションはテキストタイプまたはピクチャタイプのデータをペーストボードにカットおよびコピーします。これは、ほとんどのアプリケーションでこの2つの標準データタイプが認識されているためです。ただし、アプリケーションは1つのデータを複数の異なるインスタンスのフォーマットでペーストボードに追加することができます。例えば、スプレッドシートの一部をカットまたはコピーするたびに、スプレッドシートアプリケーションはそのデータをSYLKフォーマットやTEXTフォーマットの他に、仮想的なSPSHフォーマットでも追加することができます。SPSHインスタンスにはアプリケーションのデータ構造を使用してフォーマットされたデータが含まれています。SYLK形式には同じデータが含まれていますが、SYLKフォーマットを用いると、他の多くのスプレッドシートプログラムからも認識されます。最後に、TEXTフォーマットには同じデータが含まれますが、SYLKやSPSHフォーマットに含まれる追加のな情報は入っていません。この点で、4Dとその仮想的なスプレッドシートアプリケーション間でのカット/コピー/ペーストルーチンを記述する際に、SPSHフォーマットの内容を知り、SYLKデータの解析準備ができた場合には、以下のようなコードを作成することができます:

```
Case of
  ` まずペーストボードに仮想的なスプレッドシートのデータが含まれるかチェック
   :(Pasteboard data size("SPSH")>0)
  ` ...
  ` 次にペーストボードにSylkデータが含まれるかチェック
   :(Pasteboard data size("SYLK")>0)
  ` ...
  ` 次にペーストボードにTextデータが含まれるかチェック
   :(Pasteboard data size("TEXT")>0)
  ` ...
End case
```

つまり、オリジナルの情報の大部分を含むデータのインスタンスをペーストボードから取り出そうとしています。

例題 3

APPEND DATA TO PASTEBOARD コマンドの例題参照

SET FILE TO PASTEBOARD

SET FILE TO PASTEBOARD (filePath {; *})

引数	型		説明
filePath	文字	⇒	ファイルの完全パス名
*	演算子	⇒	指定時: 追加、省略時: 置換

説明

SET FILE TO PASTEBOARD コマンドはfilePath 引数に渡した完全パス名を追加します。このコマンドは例えば、4Dオブジェクトをデスクトップ上のファイルにドラッグ＆ドロップさせるインタフェースのセットアップに使用できます。

filePath引数には完全パス名または単に (パス名なしの) ファイル名を渡すことができます。後者の場合、ファイルはストラクチャファイルと同階層になければなりません。

コマンドはアスタリスク * をオプションの引数として受け入れます。この引数が省略されるとデフォルトで、コマンドはペーストボードの内容をfilePathで指定された最後のパス名で置き換えます。この引数を渡すと、コマンドはfilePathをペーストボードに追加します。この方法で、パス名のスタックをペーストボードに格納できます。両ケースで、ペーストボード内にパス名以外のデータが存在すると、それは消去されます。

Note: On Drag Over フォームイベント中ペーストボードは読み込みのみモードです。このコンテキストではこのコマンドは使用できません。

SET PICTURE TO PASTEBOARD

SET PICTURE TO PASTEBOARD (picture)

引数	型		説明
picture	ピクチャー	→	ペーストボードに置くピクチャ

説明

SET PICTURE TO PASTEBOARD は、ペーストボードを消去し、pictureに渡したピクチャのコピーをペーストボードに置きます。

Note: コピー/ペースト操作では、ペーストボードはクリップボードと同じです。

ピクチャは (jpeg, tif, png等の) ネイティブフォーマットで転送されます。

ペーストボードにピクチャを置いた後、GET PICTURE FROM PASTEBOARD コマンド または例えば GET PASTEBOARD DATA(**"com.4d.private.picture.gif";...)** を使用して、そのピクチャを取り出すことができます。

例題

以下の例はフローティングウィンドウを使用して、配列asEmployeeNameを含むフォームを表示します。この配列は[Employees]テーブルの従業員名を一覧表示したものです。従業員の名字をクリックするたびに、その従業員の写真をペーストボードにコピーします。この配列用のオブジェクトメソッドを以下に示します:

```
If(asEmployeeName#0)
   QUERY([Employees];[Employees]Last name=asEmployeeName{asEmployeeName})
   If(Picture size([Employees]Photo)>0)
      SET PICTURE TO PASTEBOARD([Employees]Photo) ` Copy the employee's photo
   Else
      CLEAR PASTEBOARD ` No photo or no record found
   End if
End if
```

システム変数およびセット

ピクチャのコピーが正しくペーストボードに置かれると、OK変数は1に設定されます。
ペーストボードにピクチャを置くためのメモリが十分にない場合、OK変数は0に設定されますが、エラーは生成されません。

SET TEXT TO PASTEBOARD

SET TEXT TO PASTEBOARD (text)
引数型　　説明
text　文字 → ペーストボードに置くテキスト

説明

SET TEXT TO PASTEBOARD は、ペーストボードを消去し、textに渡したテキストのコピーをペーストボードに置きます。

Note: コピー/ペースト操作では、ペーストボードはクリップボードと同じです。

ペーストボードにテキストを置いた後、Get text from pasteboard コマンド または例えば GET PASTEBOARD DATA **("com.4d.private.text.native";...)** を使用して、そのテキストを取り出すことができます。

4Dのテキスト式は最大2 GBのテキストを含めることができます。

Note: On Drag Over フォームイベント中ペーストボードは読み込みのみモードです。このコンテキストではこのコマンドは使用できません。

例題

APPEND DATA TO PASTEBOARD コマンドの例題参照

システム変数およびセット

テキストのコピーが正しくペーストボードに置かれると、OK変数は1に設定されます。
ペーストボードにテキストを置くためのメモリが十分にない場合、OK変数は0に設定されますが、エラーは生成されません。

メッセージ

- ALERT
- CONFIRM
- DISPLAY NOTIFICATION Updated 14.0
- GOTO XY
- MESSAGE
- MESSAGES OFF
- MESSAGES ON
- Request

ALERT

ALERT (message {; OK button title})

引数	型	説明
message	文字 →	アラートダイアログボックスに表示するメッセージ
OK button title	文字 →	OKボタンのタイトル

説明

ALERT コマンドは、注意アイコンとメッセージ、OKボタンで構成される警告ダイアログボックスを表示します。

message引数には表示するメッセージを渡します。このメッセージは最大255バイトまで指定可能ですが、表示されるメッセージがメッセージエリアに表示しきれない場合は、文字の幅にもよりますが途中で切り取られます。

デフォルトでは、OKボタンのタイトルは"OK"です。OKボタンタイトルを変更するには、オプションのok button title引数に新しいカスタムボタンタイトルを指定します。必要であれば、渡したカスタムタイトルの幅に応じて、OKボタンの幅を左方向にリサイズします。

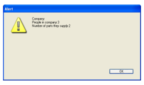
Tip: フォームあるいはオブジェクトメソッド中で、On Activate や On Deactivateを処理するセクションからALERTコマンドを呼ばないでください。これは永久ループを引き起こします。

例題 1

この例は、会社に関する情報を示すアラートボックスを表示します。表示する文字列中にキャリッジリターンが含まれてることに注目してください。これは、キャリッジリターンから後ろの文字列を次の行に改行するためです:

```
ALERT("Company: "+[Companies]Name+"\rPeople in company: "+
String(Records in selection([People]))+"\rNumber of parts they supply: "+
String(Records in selection([Parts])))
```

このコードは以下のようなアラートボックスを表示します:

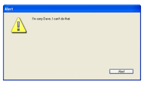

Company:
People in company:3
Number of parts they supply:2

例題 2

以下の行は:

```
ALERT("I'm sorry Dave, I can't do that.";"Alas!")
```

以下のようなアラートダイアログボックスを表示します:

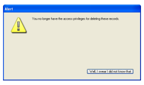

I'm sorry Dave, I can't do that.

例題 3

以下のコードは:

```
ALERT("You no longer have the access privileges for deleting these records.";"Well, I swear I did not know that")
```

以下のようなアラートダイアログボックスを表示します:


You no longer have the access privileges for deleting these records.

CONFIRM

CONFIRM (message {; OK button title {; cancel button title}})

引数	型		説明
message	文字	→	確認ダイアログボックスに表示するメッセージ
OK button title	文字	→	OKボタンのタイトル
cancel button title	文字	→	キャンセルボタンのタイトル

説明

CONFIRM コマンドは、注意アイコンとメッセージ、OKボタン、キャンセルボタンで構成される確認ダイアログボックスを表示します。

message引数には表示するメッセージを渡します。このメッセージは最大255バイトまで指定可能ですが、表示されるメッセージがメッセージエリアに表示しきれない場合は、文字の幅にもよりますが途中で切り取られます。

デフォルトでは、OKボタンのタイトルは"OK"で、キャンセルボタンのタイトルは"キャンセル"です。これらのボタンタイトルを変更するには、オプションのok button titleやcancel button title引数に新しいカスタムボタンタイトルを指定します。必要であれば、渡したカスタムタイトルの幅に応じて、ボタンの幅を左方向にリサイズします。

OKボタンはデフォルトボタンです。ユーザがOKボタンをクリックするかEnterキーを押してダイアログを受け入れると、OKシステム変数が1に設定されます。ユーザがキャンセルボタンをクリックしてダイアログをキャンセルすると、OKシステム変数は0に設定されます。

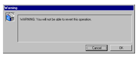
Tip: フォームあるいはオブジェクトメソッド中で、On Activate や On Deactivateを処理するセクションからCONFIRMコマンドを呼ばないでください。これは永久ループを引き起こします。

例題 1

以下のコードは:

```
CONFIRM("Warning: You will not be able to revert this operation.")
If(OK=1)
   ALL RECORDS([Old Stuff])
   DELETE SELECTION([Old Stuff])
Else
   ALERT("Operation canceled.")
End if
```

以下のような確認ダイアログボックスを表示します:



WARNING: You will not be able to revert this operation.

例題 2

このコードは:

```
CONFIRM("Do you really want to close this account?";"Yes";"No")
```

以下のような確認ダイアログボックスを表示します:

例題 3

国際的なマーケットを考慮した4Dアプリケーションを書くとします。英語から正しくローカライズされたテキストを返すプロジェクトメソッドを用意します。配列名<>asLocalizedUIMessagesには、一般的に使用されるローカライズされたテキストが格納されています:

```
CONFIRM(INTL Text("Do you want to add a new Memo?");<>asLocalizedUIMessages{kLoc_YES};
<>asLocalizedUIMessages{kLoc_NO})
```

は以下のようなフランス語の確認ダイアログを表示することができます:

例題 4

このコードは:

```
CONFIRM("Warning: If your pursue this operation, some records will be "+"irremediably affected."+Char(13)+
"What do you want to do?";"Do NOT continue";"Continue")
```

以下のような確認ダイアログボックスを表示します:

DISPLAY NOTIFICATION

DISPLAY NOTIFICATION (title ; text {; duration})

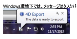
引数	型		説明
title	文字	→	通知タイトル
text	文字	→	通知テキスト
duration	倍長整数	→	表示時間 (秒)

説明

DISPLAY NOTIFICATION コマンドは ユーザーへの通知メッセージを表示します。

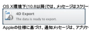
通常、このタイプのメッセージは、OSやアプリケーションがユーザに外部イベント (ネットワーク切断、アップグレードの提供等) を知らせるために使用されます。

- Windows環境下では、メッセージはタスクバーの通知領域に表示されます:



4D Export
The data is ready to export.

- OS X環境下(10.8以降)では、メッセージはスクリーンの右上隅にスライドしてくる小さなウィンドウ内に表示されます。



4D Export
The data is ready to export.

Appleの仕様に基づけ、通知メッセージは、アプリケーションが前面にない場合にのみ表示されるという点に注意して下さい。しかしながら、メッセージは通知センターのには表示されます。

titleとtextには、表示するメッセージのタイトルとテキストを渡します (上記の例では、タイトルは"4D Export"です)。255桁までの文字を入力することができます。

Windows 環境下では、メッセージウィンドウは何らかの動作がマシンで確認できるか、ユーザーが閉じるボタンをクリックするまで表示され続けます。任意の duration 引数を使用するとデフォルトの表示時間を設定することができます。通知の表示はシステム環境設定によるという点に注意して下さい。

例題

```
DISPLAY NOTIFICATION("4D Export";"The data is ready to export.")
```

GOTO XY

GOTO XY (x ; y)

引数型			説明
x	倍長整数	⇒	x カーソルの水平位置
y	倍長整数	⇒	y カーソルの垂直位置

説明

GOTO XYコマンドは、Open windowで開いたウインドウにMESSAGEコマンドでメッセージを表示する際に使用できます。

GOTO XYは、文字カーソル（見えないカーソル）の位置を指定して、ウインドウに表示される以降のメッセージの位置を設定します。

ウインドウの左上隅の位置が0,0です。カーソルは、ウインドウを開いたときと、ERASE WINDOWコマンドを実行した後には、自動的に0.0に置かれます。

GOTO XYコマンドでカーソルの位置を指定してから、MESSAGEコマンドでウインドウに文字を表示することができます。

例題 1

MESSAGEコマンドの例題参照

例題 2

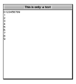
Millisecondsコマンドの例題参照

例題 3

以下の例題は:

```
Open window(50;50;300;300;5;"This is only a test")
For($vlRow;0;9)
   GOTO XY($vlRow;0)
   MESSAGE(String($vlRow))
End for
For($vlLine;0;9)
   GOTO XY(0;$vlLine)
   MESSAGE(String($vlLine))
End for
$vhStartTime:=Current time
Repeat
Until((Current time-$vhStartTime)>?00:00:30?)
```

30秒間以下のようなダイアログを表示します:



```
0123456789
1
2
3
4
5
6
7
8
9
```

MESSAGE

MESSAGE (message)

引数	型		説明
message	文字	→	表示するメッセージ

説明

MESSAGE コマンドは、通常ユーザに対して何らかの動作を知らせるために使用します。このコマンドは画面上の特別なメッセージウィンドウにmessage を表示します。このメッセージウィンドウは、Open window を使って（後述）あらかじめ開かれたウィンドウを使用していないかぎり、MESSAGE コマンドをコールするたびに表示されたり閉じられたりします。このメッセージは一時的なもので、フォームを表示する、またはメソッドの実行が終了するとすぐに消去されます。別のMESSAGE コマンドを実行すると古いメッセージは、消去されます。

Open window でウィンドウを開いている場合、続く MESSAGE コマンドの呼び出しはすべてそのウィンドウにメッセージを表示します。ウィンドウはターミナルのようなに振舞います:

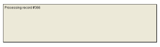
- 一連のメッセージがこのウィンドウで表示されると、前のメッセージを消去しません。その代わりに、新しいメッセージは既存のメッセージに続けて表示されます。
- メッセージがウィンドウの幅よりも長い場合、4Dは自動的に改行を行います。
- メッセージの行がウィンドウの高さより高い場合、4Dは自動的にメッセージウィンドウをスクロールします。
- 行の制御を行うには、メッセージ中にキャリッジリターン (Char(13) または "\r") を挿入します。
- ウィンドウの特定の位置にメッセージを表示するには、GOTO XYコマンドを使用します。
- ウィンドウの内容を消去するには、ERASE WINDOWコマンドを使用します。
- このウィンドウは単なる出力用ウィンドウであり、別のウィンドウがオーバーラップしても再描画されません。
- ウィンドウに表示される文字のフォントとサイズは、データベース設定"インターフェース"ページ内にて変更することができます。

例題 1

以下の例は、レコードセレクションを処理し、MESSAGEコマンドをコールしてユーザに処理の進捗状況を知らせます:

```
For($vlRecord;1;Records in selection([anyTable]))
   MESSAGE("Processing record #"+String($vlRecord))
 ` Do Something with the record
   NEXT RECORD([anyTable])
End for
```

MESSAGEをコールするたびに、以下のウィンドウが表示されては消えます:



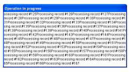
```
Processing record #366
```

例題 2

ウィンドウのちらつきを避けるため、以下の例題のようにOpen windowで開いたウィンドウにメッセージを表示することができます:

```
Open window(50;50;500;250;5;"Operation in Progress")
For($vlRecord;1;Records in selection([anyTable]))
   MESSAGE("Processing record #"+String($vlRecord))
 ` Do Something with the record
   NEXT RECORD([anyTable])
End for
CLOSE WINDOW
```

結果は以下の通り:



```
ocessing record #125Processing record #126Processing record #127Processing
record #128Processing record #129Processing record #130Processing record #1
31Processing record #132Processing record #133Processing record #134Proces
sing record #135Processing record #136Processing record #137Processing reco
rd #138Processing record #139Processing record #140Processing record #141Pr
ocessing record #142Processing record #143Processing record #144Processing
record #145Processing record #146Processing record #147Processing record #1
48Processing record #149Processing record #150Processing record #151Proces
sing record #152Processing record #153Processing record #154Processing reco
rd #155Processing record #156Processing record #157Processing record #158Pr
ocessing record #159Processing record #160Processing record #161Processing
record #162Processing record #163Processing record #164Processing record #1
65Processing record #166Processing record #167
```

例題 3

改行を追加し、見やすくします:

```
Open window(50;50;500;250;5;"Operation in Progress")
For($vlRecord;1;Records in selection([anyTable]))
   MESSAGE("Processing record #"+String($vlRecord)+Char(Carriage return))
 ` Do Something with the record
   NEXT RECORD([anyTable])
End for
CLOSE WINDOW
```

結果は以下の通り:

例題 4

GOTO XYコマンドを使用し、何行か追加します:

```
Open window(50;50;500;250;5;"Operation in Progress")
$vlNbRecords:=Records in selection([anyTable])
$vhStartTime:=Current time
For($vlRecord;1;$vlNbRecords)
   GOTO XY(5;2)
   MESSAGE("Processing record #"+String($vlRecord)+Char(Carriage return))
 ` Do Something with the record
   NEXT RECORD([anyTable])
   GOTO XY(5;5)
   $vlRemaining:=(($vlNbRecords/$vlRecord)-1)*(Current time-$vhStartTime)
   MESSAGE("Estimated remaining 時間: "+Time string($vlRemaining))
End for
CLOSE WINDOW
```

結果は以下の通り:

MESSAGES OFF

MESSAGES OFF
このコマンドは引数を必要としません

説明

MESSAGES ONとMESSAGES OFFコマンドは、時間のかかる処理を行っている際に4Dが表示する進捗インジケータの表示/非表示を切り替えます。デフォルトでは、メッセージは表示されます。

進捗インジケータを表示する処理を以下の表に示します:

フォーミュラの適用	クイックレポート	並び替え
データの書き出し	データの読み込み	グラフ
フォームによるクエリ	フォーミュラによるクエリ	クエリエディタ

進捗インジケータを表示するコマンドを以下の表に示します:

APPLY TO SELECTION
Average
BUILD APPLICATION
DISTINCT VALUES
EXPORT DIF
EXPORT SYLK
EXPORT TEXT
_o_GRAPH TABLE
IMPORT DIF
IMPORT SYLK
IMPORT TEXT
Max
Min
ORDER BY
ORDER BY FORMULA
QR REPORT
QUERY
QUERY BY FORMULA
QUERY BY EXAMPLE
QUERY SELECTION
QUERY SELECTION BY FORMULA
REDUCE SELECTION
RELATE MANY SELECTION
RELATE ONE SELECTION
SCAN INDEX
Sum

例題

以下の例は、並び替えを実行する前に進捗インジケータを非表示にし、処理が完了した時点で表示に戻します:

```
MESSAGES OFF
ORDER BY([Addresses];[Addresses]ZIP;>;[Addresses]Name2;>)
MESSAGES ON
```

MESSAGES ON

MESSAGES ON
このコマンドは引数を必要としません

説明

MESSAGES OFF コマンドの説明を参照してください。

Request

Request (message {; defaultResponse {; OKButtonTitle {; CancelButtonTitle}}}) -> 戻り値

引数	型		説明
message	文字	→	リクエストダイアログボックスに表示するメッセージ
defaultResponse	文字	→	テキスト入力エリアにデフォルトで表示するデータ
OKButtonTitle	文字	→	OKボタンのタイトル
CancelButtonTitle	文字	→	キャンセルボタンのタイトル
戻り値	文字	↩	ユーザが入力した値

説明

Requestコマンドは、メッセージ、テキスト入力エリア、OKボタン、キャンセルボタンで構成されるリクエストダイアログボックスを表示します。

message引数には表示するメッセージを渡します。メッセージが表示エリアに収まりきらない場合 (通常50文字程度ですが、使用される文字種やOS、言語により異なります) は切り取られます。

デフォルトでは、**OK**ボタンのタイトルは"OK"で、**キャンセル**ボタンのタイトルは"キャンセル"です。これらのボタンタイトルを変更するには、オプションのOKButtonTitleやCancelButtonTitle引数に新しいカスタムボタンタイトルを指定します。必要であれば、渡したカスタムタイトルの幅に応じて、ボタンの幅を左方向にリサイズします。

OKボタンはデフォルトボタンです。ユーザが**OK**ボタンをクリックするか**Enter**キーを押してダイアログを受け入れると、OKシステム変数が1に設定されます。ユーザが**キャンセル**ボタンをクリックしてダイアログをキャンセルすると、OKシステム変数は0に設定されます。

ユーザはテキスト入力エリアにテキストを入力できます。デフォルトの値を指定するには、defaultResponse引数にデフォルトのテキストを渡します。ユーザが**OK**をクリックすると**Request**はその文字列を返します。ユーザが**キャンセル**をクリックすると**Request**は空の文字列 ("") を返します。返される値が数値または日付のいずれかでなければならない場合は、**Request**が返した文字列に対してNumやDateを使用して正しいデータタイプに変換します。

Tip: フォームあるいはオブジェクトメソッド中で、On Activate や On Deactivateを処理するセクションから**Request**コマンドを呼ばないでください。これは永久ループを引き起こします。

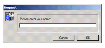
Tip: ユーザから複数の情報を得なければならない場合、**Request**ダイアログボックスを何度も表示するのではなく、そのためのフォームを作成してDIALOGでそれを表示します。

例題 1

このコードは:

```
$vsPrompt:=Request("Please enter your name:")
```

以下のようなリクエストダイアログボックスを表示します:



例題 2

このコードは:

```
vsPrompt:=Request("Name of the Employee:";"";"Create Record";"Cancel")
```

以下のようなリクエストダイアログボックスを表示します:

例題 3

このコードは:

```
$vdPrompt:=Date(Request("Enter the new date:";String(Current date)))
```

以下のようなリクエストダイアログボックスを表示します:

メニュー

メニューの管理
APPEND MENU ITEM
Count menu items
Count menus
Create menu
DELETE MENU ITEM
DISABLE MENU ITEM
Dynamic pop up menu
ENABLE MENU ITEM
Get menu bar reference
Get menu item
GET MENU ITEM ICON
Get menu item key
Get menu item mark
Get menu item method
Get menu item modifiers
Get menu item parameter
GET MENU ITEM PROPERTY
Get menu item style
GET MENU ITEMS
Get menu title
Get selected menu item parameter
INSERT MENU ITEM
Menu selected
RELEASE MENU
SET MENU BAR
SET MENU ITEM
SET MENU ITEM ICON
SET MENU ITEM MARK
SET MENU ITEM METHOD
SET MENU ITEM PARAMETER
SET MENU ITEM PROPERTY
SET MENU ITEM SHORTCUT
SET MENU ITEM STYLE

メニューの管理

用語

メニューコマンドに関する説明では、メニューの行について述べる部分で**メニューコマンド**と**メニュー項目**という用語が同等の意味で使用されています。

MenuRefと メニュー番号

4Dランゲージではメニューやメニューバーの管理に、参照によるものと番号によるもの、2つの方法を使用できます。

- 参照 (MenuRef) を使用したメニューの管理は、4D v11で導入された新しいメニュー管理の方法です。このモードは、完全に動的な (メニューエディタに存在する必要なく、オンザフライで作成されたメニュー) インタフェースの作成や、マルチレベルの階層サブメニュー管理のような上級機能へのアクセスを提供します。
- 番号によるメニューやメニューバーの管理は、デザインモードのメニューエディタに作成されたメニューに基づきます。(エディタ中の位置に対応する) メニューバーやメニューごとに固定の番号が割り当てられます。この番号はメニューやメニューバーを指定するためにコマンドで使用されます。番号により管理されるメニューに適用するランゲージコマンドのスコープは、カレントメニューバーです。

  この動作は以前のバージョンの4Dに対応し、後ほど"番号によるメニューの管理"で説明するいくつかのルールに従います。この方法はまだ使用できますが、バージョン11から提供される新しい機能 (特に動的なメニュー管理と階層サブメニュー) を使用できません。番号を使用した場合、階層サブメニューにはアクセスできません。

両方のメニュー管理モードは相互に互換性があり、あなたのインタフェースで同時に使用できます。"メニュー"テーマのほとんどのコマンドは、メニュー番号と参照両方をサポートします。
しかし、より多くの機能を使用できるため、参照によるメニュー管理をお勧めします。メニューインタフェースが部分的あるいは完全にメニューエディタで定義されているなら、Get menu bar reference や GET MENU ITEMSコマンドを使用して、参照による形式で完全に管理できることに留意してください。

参照によるメニュー管理

メニューがMenuRef参照を使用して処理される場合、メニューとメニューバーの間に違いはありません。両方とも項目のリストから構成されます。それらの利用方法のみが異なります。メニューバーの場合、メニューに対応するそれぞれの項目は、それ自身が項目を構成します。これは階層メニューベースのメニューでも同じ原則です。項目はそれ自身がメニューとなり得ます。

メニューが参照で管理される場合、セッション中でこのメニューに対して行われた変更は、このメニューのすべてのインスタンスおよびデータベースのすべてのプロセスに渡されます。

MenuRef

階層リストのように、すべての項目は、セッション全体を通して識別できるようにするユニークな参照を持ちます。この参照はMenuRefと呼ばれ、16文字で表されます。"メニュー"テーマのすべてのコマンドは、メニューを指定するために参照とメニュー番号両方を受け入れます。

メニュー参照はCreate menu, Get menu bar reference または GET MENU ITEMSコマンドを使用して取得できます。

番号によるメニュー管理

メニューバー

メニューバーは、デザインモードのメニューエディタで定義できます。番号で管理する場合、それぞれのメニューバーは番号または名前で識別されます。(4Dが自動で作成する) 最初のメニューバーは番号が1で、名前はデフォルトでメニューバー番号1となります。メニューエディタ内でこの名前を変更できます。メニューバー名は31 文字までを使用でき、ユニークでなければなりません。

1番目のメニューバーはデフォルトのメニューバーでもあります。1番目のメニューバー以外でアプリケーションを開くには、でSET MENU BAR コマンドを使用します。メニューバーの内容をプログラムで変更することはできません。しかし、それを含むメニューを変更することはできます。静的なメニューに適用されたランゲージコマンドのスコープは、カレントメニューバーです。(* 引数なしの) SET MENU BAR コマンドの呼び出しごとに、すべてのメニューとメニューコマンドは、メニューエディタで定義された元の状態に戻ります。

すべてのメニューバーは、3つのメニュー、ファイル、編集、およびモードメニューを持ちます。

- ファイルメニューには終了コマンドしかありません。終了標準アクションが割り当てられ、このアクションは「終了しますか?」のダイアログボックスを表示し、4Dアプリケーションの終了または取り消しを行うことができます。

Note: Mac OS Xでは、メニューコマンドの終了に関連づけられるアクションは、自動的にデータベースを起動しているシステムのアプリケーションメニューに配置されます。

ファイルメニュー名を変更したり、メニューコマンドを追加したり、あるいはそのまま使用することも出来ます。ファイルメニューの最後の項目を終了とすることをお勧めします。

- 編集メニューには、標準的な編集メニューコマンドが含まれます。標準アクション(取り消し、カット、コピー、ペースト等)がメニュー項目に割り付けられています。開発者は、コマンドを追加したり、独自のメソッドにより編集アクションを管理することが出来ます。

- モードメニューには、デザインモードに戻るコマンドが含まれます。このコマンドは、デザインモードが有効であれば、アプリケーションモードからそこに戻るために使用されます。

Note: 4DはヘルプとMac OS Xのアプリケーションシステムメニューを自動的に管理します。このメニューは4Dについてコマンドを除き、変更することはできません。4DについてはSET ABOUTコマンドで管理できます。

警告: メニューバーは"インタープロセス"です。デザインモードで行われたメニューバー上の更新は、すべてのメニューバーを使用しているプロセスに反映されます。

メニュー番号とメニューコマンド番号

メニューバーのように、メニューにも番号が付けられています。通常ファイルメニューはメニュー1です。その後メニューは左から右に順に番号が付けられます。アプリケーションメニュー (Mac OS) はこのメニューの番号付からは外されます。両プラットフォームにおいて、ヘルプメニューも番号付からは外されます。Count menus コマンドはこれらのメニューを考慮に入れないことに留意してください。例えばメニューバーがファイル, 編集, 顧客, 請求 そして ヘルプメニューから構成されているとき、Count menus は4Dが管理するシステムメニューを除いて、4を返します。

メニューの番号付は例えばMenu selected関数を使用する際に重要です。

メニューがフォームに関連付けられている場合、メニューの番号付スキームは異なります。最初に追加されたメニューは2049から始まります。追加されたメニューを参照するには、通常のメニュー番号に2048を加えます。

各メニュー内のメニュー項目は、項目の1番上から下に向かって連続番号が付けられています (区切り線を含む)。1番上のメニューがアイテム1です。

連結メニューバー

フォームプロパティを使用してフォームにメニューバーを関連付けることができます。このようなメニューをこのドキュメントでは "フォームメニューバー" と呼びます。

フォームメニューバー上のメニューは、アプリケーションモードで、出力フォームとしてフォームが表示された際のカレントメニューバーに追加されます。

フォームメニューバーは、メニューバー番号で指定します。カレントフォームに付随して表示されたメニューバーの番号が、そのフォームに追加されたメニューバーの番号と同じである場合には、メニューバーは追加されません。
フォームメニューバーはメニューバー番号やメニュー名で指定されます。番号や名前がカレントのメニューバーと同じ場合、メニューバーは追加されません。

デフォルトで、カスタムメニューバーとともにフォームを表示した時、メニューバーコマンドは無効にされています。つまり、選択してもコマンドは実行 されません。フォームプロパティのアクティブメニューバーオプションをチェックすることによって動作を変更することができます。これにより、カレントメ ニューバーのコマンドが使用可能となります。
すべての場合において、メニュー コマンドの選択はフォームメソッドにOn Menu Selectedイベントを発生させます。そこで、選ばれたメニューをテストするためにMenu Selectedコマンドを使うことができます。

添付メニュー

メニューをメニューバーに添付できます。添付されたメニューがコマンドで変更されると、すべてのインスタンスにこの変更が反映されます。添付メニューに関する詳細は、4D Design Reference マニュアルを参照してください。

メニューコマンドに関連付けられた標準アクションとメソッド

各メニューコマンドには、プロジェクトメソッドまたは標準アクションを持つことができます。メニューにメソッドまたは標準アクションを割り当てていないと、そのメニューを選択したとき4Dはアプリケーションモードから抜けてデザインモードへ戻ります。アプリケーションモードだけが利用可能な場合や、ユーザがデザインモードへのアクセス権を持たない場合、4Dは終了します。
標準アクションは、システム機能(コピー、終了等)に、または4Dのデータベース(レコード追加、すべてを選択等)にリンクされた様々な基本操作を実行するために使用できます。
標準アクションとプロジェクトメソッドの両方をメニューコマンドに割り当てることができます。この場合、標準アクションは実行されません。しかし現在のコンテキストに従って4Dは、メニュー項目を選択可/選択不可に変更したり、プラットフォーム特有のメニュー処理 (例えばMac OSで環境設定メニューをアプリケーションメニューに移動する) などを行ったりします。メニュー項目が選択不可になった場合、関連づけられたプロジェクトメソッドは実行されません。

menuItem=-1

メニュー項目の管理を容易にするために、4Dはメニューに最後に追加された項目を指定するためのショートカットを提供します。この場合、menuItem引数に-1を渡します。
この原則は"メニュー"テーマのメニュー項目を扱うすべてのコマンドに適用できます。

APPEND MENU ITEM

APPEND MENU ITEM (menu ; itemText {; subMenu {; process {; *}}})

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー番号またはメニュー参照
itemText	テキスト	→	新しいメニュー項目のテキスト
subMenu	MenuRef	→	項目に関連付けるサブメニューの参照
process	倍長整数	→	プロセス参照番号
*	演算子	→	指定時: メタ文字を標準文字として扱う

説明

APPEND MENU ITEM コマンドは、menu引数に渡されたメニュー番号または参照を持つメニューに新規メニュー項目を追加します。

process引数を省略すると、APPEND MENU ITEMコマンドはカレントプロセスのメニューバーに適用されます。そうでない場合、APPEND MENU ITEMはprocessに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡すと、process引数は意味を持たず、無視されます。

* 引数を渡さない場合、APPEND MENU ITEMコマンドは、1回の呼び出しで1つまたは複数のメニュー項目を追加することができます。

追加されるメニュー項目を以下のようにしてitemText引数で定義します:

- セミコロン (;) で各メニューを区切る。例えば、
"ItemText1;ItemText2;ItemText3"
- メニュー項目を使用不可にする場合は、項目テキストに開き丸カッコ (() を配置する。
- メニュー項目の区切り線を指定する場合は、項目テキストに"-" または "(-" を渡す。
- 行にフォントスタイルを指定する場合は、項目テキストに小なり記号 (<) の後に下記の文字の1つを配置する:
  <B 太字
  <I イタリック
  <U 下線
- メニュー項目にチェックマークを追加するには、項目テキストにエクスクラメーションマーク (!) を指定し、その後にチェックマークとして使用する文字を配置する。Macintosh上ではその配置された文字が表示されます。Windows上ではどの文字が渡されようともチェックマークが表示される。
- メニュー項目にアイコンを追加するには、項目テキストにアクセント記号 (^) を指定し、ASCIIコードから208を引いた値がMacintoshベースのリソースIDである文字をその後に配置する。
- 任意のメニュー項目にキーボードショートカットを追加するには、項目テキストにスラッシュ (/) を指定し、その後にショートカット用の文字を配置する。

Note: メニュー項目の数が程よいメニューを使用してください。例えばもし50以上のメニュー項目を表示したい場合は、メニューの代わりにフォーム中にスクロールエリアを使用することを検討してみてください。

* 引数を渡すと、項目テキストに含められた特別文字 (; (!...) は標準の文字として扱われ、メタ文字ではなくなります。つまり"コピー (特別)..."や"検索/置換..."のようなメニュー項目を作成できるようになります。* 引数が渡されたときは、一回のコマンドの呼び出しで複数の項目を作成できないことに注意してください。この場合 ";" 文字は標準の文字として扱われます。

Note: GET MENU ITEMSとGet menu itemコマンドはメニュー項目が作られた方法により、メタ文字を返したり返さなかったりします。* オプション付きで作成された場合、メタ文字は標準文字として返されます。

オプションのsubMenu引数を使用して、階層サブメニューとして追加するメニューを指定できます。Create menuコマンド等を使用して作成されたメニュー参照 (MenuRef 型文字列) を渡さなければなりません。コマンドが2つ以上のメニュー項目を追加する場合、サブメニューは最初の項目に追加されます。

重要: 新しいメニュー項目にはメソッドやアクションが割り当てられていません。これらはSET MENU ITEM PROPERTYやSET MENU ITEM METHODコマンドを使用して項目に割り当てるか、Menu selectedコマンドを使ってフォームメソッドからこれを管理しなくてはなりません。

例題

この例題ではカレントメニューバー6番目の項目のFontメニューに、利用可能なフォント名を追加します:

```
   ` On Startup データベースメソッドで
   ` フォントリストがロードされ、メニュー項目テキストが構築される
 FONT LIST(<>asAvailableFont)
 <>atFontMenuItems:=""
 For($vlFont;1;Size of array(<>asAvailableFont))
    <>atFontMenuItems:=<>atFontMenuItems+";"+<>asAvailableFont{$vlFont}
 End for
```

フォームメソッドやプロジェクトメソッドで、以下のように記述できます:

```
 APPEND MENU ITEM(6;<>atFontMenuItems)
```

Count menu items

Count menu items (menu {; process}) -> 戻り値

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー番号またはメニュー参照
process	倍長整数	→	プロセス参照番号
戻り値	倍長整数	↩	メニュー中のメニュー項目数

説明

Count menu items コマンドは、menu引数に渡されたメニュー番号または参照を持つメニュー内にあるメニュー項目の数を返します。

process 引数を省略すると、Count menu items はカレントプロセスのメニューに適用されます。そうでない場合、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡すと、process 引数は意味を持たず、無視されます。

Count menus

Count menus {(process)} -> 戻り値

引数	型		説明
process	倍長整数	→	プロセス参照番号
戻り値	倍長整数	↩	カレントメニューバー中のメニュー数

説明

Count menus コマンドは、メニューバー上にあるメニューの数を返します。

process引数を省略すると、Count menusコマンドはカレントプロセスのメニューバーに適用されます。そうでなければ、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Create menu

Create menu {(menu)} -> 戻り値

引数	型		説明
menu	MenuRef, 倍長整数, 文字	→	メニュー参照 または 番号 または メニューバー名
戻り値	MenuRef	↩	メニュー参照

説明

Create menu コマンドはメモリに新しいメニュを作成するために使用します。このメニューはメモリ上にのみ存在し、デザインモードのメニューバーには追加されません。セッション中にこのメニューに対して行われた更新は、データベースのすべてのプロセスの、このメニューすべてのインスタンスに即座に反映されます。

コマンドは新しいメニューのMenuRef型のIDを返します。

- オプションのmenu引数を渡さない場合、空のメニューが作成されます。APPEND MENU ITEMやSET MENU ITEMなどのコマンドを使用して、メニューを構築、管理しなくてはなりません。
- menu 引数を渡すと、作成されるメニューはソースメニューの完全なコピーとなります。サブメニューを含む、ソースメニューのすべてのプロパティが、新しいメニューに適用されます。新しいMenuRef参照がソースメニューと既存のサブメニューに作成されることに注意してください。

menu 引数には、有効なメニュー参照、またはデザインモードで定義したメニュー名やメニュー番号を渡すことができます。最後のケースでは、新しいメニューはソースメニューバーのメニューやサブメニューで構成されます。

このコマンドで作成されたメニューは、SET MENU BAR コマンドでメニューバーとして使用できます。

Create menuで作成したメニューが必要なくなったときには、RELEASE MENUコマンドを呼び出して使用されているメモリを解放してください。

例題

SET MENU BAR コマンドの例題を参照。

DELETE MENU ITEM

DELETE MENU ITEM (menu ; menuItem {; process})

引数	型		説明
menu	倍長整数, MenuRef	⇒	メニュー番号またはメニュー参照
menuItem	倍長整数	⇒	メニュー項目番号 または -1: 最後に追加された項目
process	倍長整数	⇒	プロセス参照番号

説明

DELETE MENU ITEM コマンドは、menu引数にメニュー番号またはメニュー参照で指定したメニューから、menuItem引数にメニュー項目番号で指定したメニュー項目を削除します。menuItemに-1を渡すと、menuに最後に追加された項目を指定します。

menu and menuItemで指定されたメニュー項目が、Create menu コマンド等を使用して作成され、参照で管理されるメニューの場合、DELETE MENU ITEM はmenu中のmenuItem のインスタンスのみを削除します。menuItemに参照されるサブメニューはメモリ中に引き続き存在します。参照により管理されるメニューを削除するためにはRELEASE MENU コマンドを使用しなければなりません。
このコマンドはCreate menu コマンドを使用して作成され、SET MENU BAR コマンドでインストールされたメニューバーにも使用できます。

オプション引数processを省略すると、DELETE MENU ITEMコマンドはカレントプロセスのメニューバーに適用されます。processを指定した場合は、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

Note: ユーザインタフェースの一貫性を保つため、項目のないメニューを保持してはいけません。

DISABLE MENU ITEM

DISABLE MENU ITEM (menu ; menuItem {; process})

引数	型	説明
menu	倍長整数, MenuRef	→ メニュー番号またはメニュー参照
menuItem	倍長整数	→ メニュー項目番号 または -1: 最後に追加された項目
process	倍長整数	→ プロセス参照番号

説明

DISABLE MENU ITEM コマンドは、menu引数にメニュー番号またはメニュー参照で指定したメニュー中、menuItem引数にメニュー項目番号で指定したメニュー項目を選択不可にします。menuItemに-1を渡すと、menuに最後に追加された項目を指定します。

オプション引数processを省略すると、DISABLE MENU ITEMコマンドはカレントプロセスのメニューバーに適用されます。processを指定した場合は、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

menuItem 引数が階層サブメニューを指す場合、このメニューのすべての項目とサブメニューが選択不可になります。このコマンドはCreate menu コマンドを使用して作成され、SET MENU BAR コマンドでインストールされたメニューバーにも使用できます。

Note: menuにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

Tip: 一度にすべての項目の選択可/不可を切り替えるには、menuItemに0を渡します。

Dynamic pop up menu

Dynamic pop up menu (menu {; default {; xCoord ; yCoord}}) -> 戻り値

引数	型		説明
menu	MenuRef	→	メニュー参照
default	文字	→	デフォルトで選択される項目のパラメーター
xCoord	倍長整数	→	左上隅のx座標
yCoord	倍長整数	→	左上隅のy座標
戻り値	文字	↩	選択されたメニュー項目のパラメーター

説明

Dynamic pop up menuはマウスの現在位置またはオプションのxCoord と yCoord引数で指定した位置に階層ポップアップメニューを表示します。使用される階層メニューはCreate menuコマンドで作成されていなければなりません。Create menuから返される参照をmenu 引数に渡します。

インターフェースルールの標準に準拠して、通常はこのコマンドを右クリック、または一定時間以上ボタンが押され続けた時に呼び出します (例えばコンテキストメニューとして)。

オプションのdefault 引数を使用して、メニューが表示されるときにデフォルトで選択するポップアップメニューの項目を指定できます。この引数にはメニューに割り当てたカスタム文字列を渡します。この文字列は事前にSET MENU ITEM PARAMETERコマンドを使用して設定されていなければなりません。この引数を渡さないと、デフォルトで先頭の項目が選択されます。

オプションの xCoord と yCoord 引数を使用して、ポップアップメニューが表示される場所を指定できます。xCoord と yCoord 引数にはそれぞれ、メニューの左上隅の水平および垂直座標を渡します。この座標はカレントフォームのローカル座標システムをピクセル単位で渡します。両引数はペアで指定しなければなりません。片方のみを渡した場合、無視されます。

3Dボタンに関連付けられたポップアップメニューを表示したい場合、オプションの xCoord と yCoord 引数は渡しません。この場合は、カレントプラットフォームのインタフェース標準に基づき、4Dが自動でメニューの場所を計算します。

メニュー項目が選択されると、コマンドはその (SET MENU ITEM PARAMETERコマンドを使用して定義された) カスタム文字列を返します。そうでなければコマンドは空の文字列を返します。

注: 既存のPop up menuコマンド ("ユーザインタフェース" テーマ) はテキストからポップアップメニューを作成するために使用できます。

ENABLE MENU ITEM

ENABLE MENU ITEM (menu ; menuItem {; process})

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー番号またはメニュー参照
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
process	倍長整数	→	プロセス参照番号

説明

ENABLE MENU ITEM コマンドは、menu引数にメニュー番号またはメニュー参照で指定したメニュー中、menuItem引数にメニュー項目番号で指定したメニュー項目を選択可にします。menuItemに-1を渡すと、menuに最後に追加された項目を指定します。

menuItem 引数が階層サブメニューを指す場合、このメニューのすべての項目とサブメニューが選択可になります。このコマンドはCreate menu コマンドを使用して作成され、SET MENU BAR コマンドでインストールされたメニューバーにも使用できます。

オプション引数processを省略すると、ENABLE MENU ITEMコマンドはカレントプロセスのメニューバーに適用されます。processを指定した場合は、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

Tip: 一度にすべての項目の選択可/不可を切り替えるには、menuItemに0を渡します。

Get menu bar reference

Get menu bar reference {(process)} -> 戻り値

引数	型		説明
process	倍長整数	→	プロセス参照番号
戻り値	MenuRef	↩	メニューバーID

説明

Get menu bar reference コマンドはカレントのメニューバーあるいは指定されたプロセスのメニューバーのIDを返します。

メニューバーがCreate menu コマンドで作成された場合、このIDは作成されたメニューの参照IDに対応します。そうでなければ、コマンドは特定の内部IDを返します。いずれの場合も、このMenuRef IDは、このテーマの他のすべてのコマンドでメニュー参照として使用できます。

process 引数は、カレントのメニューバーIDを取得するプロセスを指定するために使用できます。この引数を省略すると、コマンドはカレントプロセスのメニューバーIDを返します。

例題

GET MENU ITEMS コマンドの例題参照

Get menu item

Get menu item (menu ; menuItem {; process}) -> 戻り値

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー番号またはメニュー参照
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
process	倍長整数	→	プロセス参照番号
戻り値	文字	↩	メニュー項目のテキスト

説明

Get menu item コマンドは、引数menuとmenuItemに渡されたメニューおよびメニュー項目番号を持つメニュー項目のテキストを返します。menuItemに-1を渡してmenuに最後に追加された項目を指定することもできます。

process引数を省略すると、Get menu itemコマンドはカレントプロセスのメニューバーに適用されます。そうでなければ、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

GET MENU ITEM ICON

GET MENU ITEM ICON (menu ; menuItem ; iconRef {; process})

引数	型		説明
menu	倍長整数, MenuRef	⇒	メニュー参照またはメニュー番号
menuItem	倍長整数	⇒	メニュー項目番号 または -1: 最後に追加された項目
iconRef	テキスト変数, 倍長整数変数	⇐	メニュー項目に関連付けられたアイコンの ピクチャ名または番号
process	倍長整数	⇒	プロセス番号

説明

GET MENU ITEM ICON コマンドはiconRef変数に、menu と menuItem引数で指定されたメニュー項目に関連付けられたアイコンの参照を返します。この参照はピクチャ名または番号です。

メニュー項目に関連付けられたアイコンは、アプリケーションのツールバーに追加されます。

menuItemに-1を渡してmenuに最後に追加された項目を指定することができます。

menuにはメニュー参照 (MenuRef) またはメニュー番号を渡します。メニュー参照を渡す場合、process 引数は必要なく、渡されても無視されます。メニュー番号を渡す場合、コマンドはカレントプロセスのメインメニューバー中の対応するメニューを考慮します。他のプロセスを指定したい場合、オプションのprocess 引数にその番号を渡します。

アイコンがライブラリピクチャを使用して指定されていた場合、コマンドはこの引数に渡された変数のタイプに応じ、ピクチャ名あるいは番号のいずれかを返します。アイコンがデータベースのResourcesフォルダに格納されたピクチャを使用して指定されていた場合、コマンドはiconRefにピクチャのパス名を返します。

メニュー項目にアイコンが割り当てられていない場合、コマンドは空値を返します。

Get menu item key

Get menu item key (menu ; menuItem {; process}) -> 戻り値

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー参照またはメニュー番号
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
process	倍長整数	→	プロセス番号
戻り値	倍長整数	↩	メニュー項目に関連付ける 標準ショートカットキーの文字コード

説明

Get menu item key コマンドは、menu引数にメニュー番号またはメニュー参照で指定したメニュー中、menuItem引数にメニュー項目番号で指定したメニュー項目の、Ctrl (Windows) または Command (Macintosh) ショートカットコードを返します。menuItemに-1を渡してmenuに最後に追加された項目を指定することもできます。

process引数を省略すると、Get menu item keyコマンドはカレントプロセスのメニューバーに適用されます。そうでなければ、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

メニュー項目に割り当てられたショートカットがないか、menuItem 引数が階層サブメニューを指す場合、Get menu item key は 0 を返します。

例題

メニュー項目に割り当てられたショートカットを取得するために、以下のようなタイプのプログラミング構造を実装すると便利です:

```
If(Get menu item key(mymenu;1)#0)
   $modifiers:=Get menu item modifiers(mymenu;1)
   Case of
      :($modifiers=Option key mask)
         ...
      :($modifiers=Shift key mask)
         ...
      :($modifiers=Option key mask+Shift key mask)
         ...
   End case
End if
```

Get menu item mark

Get menu item mark (menu ; menuItem {; process}) -> 戻り値

引数	型	説明
menu	倍長整数, MenuRef	→ メニュー参照またはメニュー番号
menuItem	倍長整数	→ メニュー項目番号 または -1: 最後に追加された項目
process	倍長整数	→ プロセス番号
戻り値	文字	↩ カレントメニュー項目のマーク

説明

Get menu item mark コマンドは、menu引数にメニュー番号またはメニュー参照で指定したメニュー中、menuItem引数にメニュー項目番号で指定したメニュー項目の、チェックマークを返します。menuItemに-1を渡してmenuに最後に追加された項目を指定することもできます。

process引数を省略すると、Get menu item markコマンドはカレントプロセスのメニューバーに適用されます。そうでなければ、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

メニュー項目にチェックマークがないか、menuItem 引数が階層サブメニューを指す場合、Get menu item mark は空の文字列を返します。

Note: Macintosh と Windows におけるチェックマークの議論については、コマンド SET MENU ITEM MARKの説明を参照してください。

例題

以下の例題は、メニュー項目のチェックマークを切り替えます:

```
SET MENU ITEM MARK($vlMenu;$vlItem;Char(18)*Num(Get menu item mark($vlMenu;$vlItem)=""))
```

Get menu item method

Get menu item method (menu ; menuItem {; process}) -> 戻り値

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー参照またはメニュー番号
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
process	倍長整数	→	プロセス番号
戻り値	文字	↩	メソッド名

説明

Get menu item method コマンドは、menuとmenuItem引数で指定されたメニュー項目に関連付けられた4Dプロジェクトメソッド名を返します。
menuItemに-1を渡してmenuに最後に追加された項目を指定することもできます。

menuにはメニュー参照 (MenuRef) またはメニュー番号を渡します。メニュー参照を渡す場合、process 引数は必要なく、渡されても無視されます。
メニュー番号を渡す場合、コマンドはカレントプロセスのメインメニューバー中の対応するメニューを考慮します。他のプロセスを指定したい場合、オプションのprocess 引数にその番号を渡します。

コマンドは4Dメソッド名を文字列 (式) で返します。メニュー項目にメソッドが割り当てられていない場合、コマンドは空の文字列を返します。

Get menu item modifiers

Get menu item modifiers (menu ; menuItem {; process}) -> 戻り値

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー参照またはメニュー番号
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
process	倍長整数	→	プロセス番号
戻り値	倍長整数	↩	メニュー項目に割り当てられたモディファイアキー

説明

Get menu item modifiers コマンドは、menuとmenuItem引数で指定したメニュー項目の、標準ショートカットに割り当てられた追加のモディファイアキーを返します。
標準ショートカットはCtrl (Windows) または Command (Macintosh) とカスタムキーの組み合わせで構成されます。標準ショートカットはSET MENU ITEM SHORTCUT と Get menu item keyコマンドを使用して管理されます。

追加のモディファイアキーはShift キーおよびOption (Mac OS) /Alt (Windows) キーです。これらのモディファイアは、すでに標準ショートカットが指定されているときにのみ使用されます。

このコマンドから返される番号は、追加のモディファイアキーのコードに対応します。キーのコードは以下のとおりです:

- **Shift**= 512
- **Option** (Mac OS) または **Alt** (Windows) = 2048

両方のキーが使用されているとき、これらの値は加算されます。

Note: 返された値は、"" テーマのShift key mask と Option key mask 定数を使用して評価できます。

メニュー項目にモディファイアが割り当てられていないばあ、コマンドは0を返します。
menuItemに-1を渡してmenuに最後に追加された項目を指定することもできます。

menuにはメニュー参照 (MenuRef) またはメニュー番号を渡します。メニュー参照を渡す場合、process 引数は必要なく、渡されても無視されます。メニュー番号を渡す場合、コマンドはカレントプロセスのメインメニューバー中の対応するメニューを考慮します。他のプロセスを指定したい場合、オプションのprocess 引数にその番号を渡します。

例題

Get menu item key コマンドの例題参照

Get menu item parameter

Get menu item parameter (menu ; menuItem) -> 戻り値

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー参照またはメニュー番号
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
戻り値	文字	⮐	メニュー項目のカスタムパラメタ

説明

Get menu item parameter コマンドは、menuとmenuItem引数で指定されるメニュー項目に割り当てられたカスタム文字列を返します。この文字列は事前にSET MENU ITEM PARAMETERコマンドで指定されていなければなりません。

GET MENU ITEM PROPERTY

GET MENU ITEM PROPERTY (menu ; menuItem ; property ; value {; process})

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー参照またはメニュー番号
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
property	文字	→	プロパティタイプ
value	式	←	プロパティ値
process	倍長整数	→	プロセス番号

説明

GET MENU ITEM PROPERTYコマンドは、menuとmenuItem引数で指定したメニュー項目の、現在のプロパティ値を返します。menuItemに-1を渡してmenuに最後に追加された項目を指定することもできます。

menuにはメニュー参照 (MenuRef) またはメニュー番号を渡します。メニュー参照を渡す場合、process 引数は必要なく、渡されても無視されます。メニュー番号を渡す場合、コマンドはカレントプロセスのメインメニューバー中の対応するメニューを考慮します。他のプロセスを指定したい場合、オプションのprocess 引数にその番号を渡します。

property 引数には、値を取得したいプロパティを渡します。“テーマの定数またはカスタムプロパティに対応する文字列を使用できます。メニュープロパティとその値に関する詳細はSET MENU ITEM PROPERTYコマンドの説明を参照してください。

Get menu item style

Get menu item style (menu ; menuItem {; process}) -> 戻り値

引数	型		説明
menu	倍長整数, MenuRef	➡	メニュー番号またはメニュー参照
menuItem	倍長整数	➡	メニュー項目番号 または -1: 最後に追加された項目
process	倍長整数	➡	プロセス参照番号
戻り値	倍長整数	↩	現在のメニュー項目スタイル

説明

Get menu item style コマンドは、menu引数にメニュー番号またはメニュー参照で指定したメニュー中、menuItem引数にメニュー項目番号で指定したメニュー項目のフォントスタイルを返します。menuItemに-1を渡してmenuに最後に追加された項目を指定することもできます。

process引数を省略すると、Get menu item styleコマンドはカレントプロセスのメニューバーに適用されます。そうでなければ、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

Get menu item style はテーマの定義済み定数 (ひとつまたは和) を返します:

定数	型	値
Plain	倍長整数	0
Bold	倍長整数	1
Italic	倍長整数	2
Underline	倍長整数	4

例題

メニューが太字で表示されるかどうかをテストするには、以下のように書きます:

```
If((Get menu item style($vlMenu;$vlItem)&Bold)#0)
  `...
End if
```

GET MENU ITEMS

GET MENU ITEMS (menu ; menuTitlesArray ; menuRefsArray)

引数	型	説明
menu	倍長整数, MenuRef	⇒ メニュー参照またはメニュー番号
menuTitlesArray	文字配列	⇐ メニュータイトル配列
menuRefsArray	文字配列	⇐ メニュー参照配列

説明

GET MENU ITEMS コマンドは、menu引数で指定したメニューまたはメニューバーのタイトルとIDをmenuTitlesArray と menuRefsArray 配列に返します。

menu 引数にはメニュー参照 (MenuRef)、メニューバー番号、またはGet menu bar reference コマンドを使用して取得したメニューバー参照を渡します。

項目にメニュー参照が割り当てられていない場合、対応する配列要素には空の文字列が返されます。

例題

カレントプロセスのメニューバーの内容を取得します:

```
ARRAY TEXT(menuTitlesArray;0)
ARRAY TEXT(menuRefsArray;0)
MenuBarRef:=Get menu bar reference(Frontmost process)
GET MENU ITEMS(MenuBarRef;menuTitlesArray;menuRefsArray)
```

Get menu title

Get menu title (menu {; process}) -> 戻り値

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー番号またはメニュー参照
process	倍長整数	→	プロセス参照番号
戻り値	文字	↩	メニューのタイトル

説明

Get menu title コマンドは、menuに渡されたメニュー番号または参照を持つメニューのタイトルを返します。

process引数を省略すると、Get menu titleコマンドはカレントプロセスのメニューバーに適用されます。そうでなければ、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

Get selected menu item parameter

Get selected menu item parameter -> 戻り値

引数	型	説明
戻り値	文字	↩ メニュー項目のカスタムパラメタ

説明

Get selected menu item parameter コマンドは、選択されたメニュー項目に割り当てられたカスタム文字列を返します。このパラメタは事前にSET MENU ITEM PARAMETERコマンドを使用してあらかじめ設定されていなければなりません。
メニュー項目が選択されていない場合、コマンドは空の文字列を返します。

INSERT MENU ITEM

INSERT MENU ITEM (menu ; afterItem ; itemText {; subMenu {; process}}{; *})

引数	型	説明
menu	倍長整数, MenuRef	⇨メニュー番号またはメニュー参照
afterItem	倍長整数	⇨メニュー項目番号
itemText	文字	⇨挿入するメニュー項目のテキスト
subMenu	MenuRef	⇨項目に割り当てるサブメニュの参照
process	倍長整数	⇨プロセス参照番号
*	演算子	⇨指定時: メタ文字を標準文字として扱う

説明

INSERT MENU ITEMコマンドは、menuに渡されたメニュー番号または参照を持つメニューにおいて、afterItemに渡された番号の既存のメニュー項目の後ろに新しいメニュー項目を挿入します。

process引数を省略すると、INSERT MENU ITEMコマンドはカレントプロセスのメニューバーに適用されます。そうでなければ、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡した場合、process引数は意味を持たず、無視されます。

* 引数を渡さない場合、INSERT MENU ITEMは一度の呼び出しで1つまたは複数のメニュー項目の挿入ができます。

INSERT MENU ITEMは、メニュー中の任意の場所に項目を挿入できるという点を除いて、APPEND MENU ITEMのように動作します。APPEND MENU ITEMは常にメニューの最後に項目を追加します。

itemTextに渡す項目や* 引数の動作の定義については、APPEND MENU ITEMコマンドの説明を参照してください。

オプションのsubMenu 引数を使用して、階層サブメニューとして追加するメニューを指定できます。Create menu コマンド等を使用して作成されたメニュー参照 (MenuRef 型文字列) を渡さなければなりません。コマンドが2つ以上のメニュー項目を追加する場合、サブメニューは最初の項目に追加されます。

重要: 新しいメニュー項目には、メソッドやアクションが割り当てられていません。これらをSET MENU ITEM PROPERTY や SET MENU ITEM METHODコマンドを使用して項目に割り当てるか、Menu selectedコマンドを使ってフォームメソッドからこれを管理しなくてはなりません。

例題

以下の例題は2つのコマンドからなるメニューを作成し、メソッドを割り当てます:

```
menuRef:=Create menu
APPEND MENU ITEM(menuRef;"Characters")
SET MENU ITEM METHOD(menuRef;1;"CharMgmtDial")
INSERT MENU ITEM(menuRef;1;"Paragraphs")
SET MENU ITEM METHOD(menuRef;2;"ParaMgmtDial")
```

Menu selected

Menu selected {(subMenu)} -> 戻り値

引数	型		説明
subMenu	MenuRef	←	選択された項目を含むメニューの参照
戻り値	倍長整数	→	選択されたメニューコマンド 上位バイトにメニュー番号 下位バイトにメニュー項目番号

説明

Menu selected はフォームが表示されているときのみ使用できます。このコマンドはメニューから選択されたメニュー項目を検出し、階層サブメニューの場合はサブメニューの参照を返します。

Tip: 可能な限り、Menu selectedを使用しないで、(負のメニューバー番号を用いた) 関連メニューバーのメニューに割り当てられたメソッドを使用してください。いずれのメニューが選択されたかを調べる必要がないため、関連メニューバーはより簡単に管理できます。

Menu selected コマンドを階層サブメニューに対して使用できます。第一レベルより下の階層のメニュー項目を選択すると、コマンドはsubMenu 引数に、選択された項目が属するサブメニューの参照 (MenuRef 型, 16文字の文字列) を返します。メニューコマンドが階層サブメニューを含まない場合、この引数には空の文字列が返されます。

Menu selected コマンドは、選択されたメニューの番号を倍長整数型で返します。選択されたメニュー番号を知るためには、返された数値を65536で割り、その結果を整数型に変換します。選択されたメニューコマンド番号を知るには、係数65536を使い、返された数値のモジューロを計算します。メニュー番号とメニューコマンド番号を計算するには以下のフォーミュラを使用します:

```
menu:=Menu selected\ 65536
menuCommand:=Menu selected% 65536
```

を使用してこれらの値を取得することもできます:

```
menu:=(Menu selected & 0xFFFF0000)>>16
menuCommand:=Menu selected & 0xFFFF
```

メニューが選択されていない場合、Menu selectedは0を返します。

例題

以下の例はSET MENU ITEM MARKコマンドのメニューとメニュー項目引数を求めるためにMenu selectedを使用しています:

```
Case of
   :(Form event=On Menu Selected)
      C_STRING(16;$refMenuIncludingItem)
      C_LONGINT($ref;$MenuNum;$MenuItemNum)
      $ref:=Menu selected($refMenuIncludingItem)
      $MenuNum:=$ref\65536
      $MenuItemNum:=$ref%65536
      SET MENU ITEM MARK($refMenuIncludingItem;$MenuItemNum;Char(18))
End case
```

Note: 項目が選択されていなければ、On Menu Selectedフォームイベントは有効にされません。メニューが、メニューバー中のメニューの一つでない場合、$refMenuIncludingItemには常に値が与えられ、$MenuNumは0となります。

RELEASE MENU

RELEASE MENU (menu)
引数　型　　　説明
menu MenuRef → メニュー参照

説明

RELEASE MENU コマンドはmenu に渡したIDを持つメニューをメモリから解放します。このメニューはCreate menu コマンドで作成されたものでなければなりません。以下のルールが適用されます: 各Create menuに対応するRELEASE MENUコマンドが呼び出されなければなりません。

このコマンドはmenuメニューのすべてのインスタンスを、すべてのプロセスのすべてのメニューバーから取り除きます。メニューが使用中のメニューバーに属する場合、引き続き使用することはできますが、変更することはできなくなります。このメニューは、それがメニューバーに現れなくなった後に、実際にメモリから取り除かれます。

このコマンドはメニューバーとして使用されるメニューに使用できます。

menuにより使用されるサブメニューが直接Create menuコマンドで作成されていた場合は、取り除かれません。この場合個々に取り除く必要があります (先のルールが適用されます)。しかしサブメニューが既存のメニューの複製である場合、RELEASE MENUで取り除くことはできません。4Dが自動で取り除きます。

例題

この例題では、このコマンドの異なる用法を示します:

```
newMenu:=Create menu
APPEND MENU ITEM(newMenu;"Test1")
subMenu:=Create menu
APPEND MENU ITEM(subMenu;"SubTest1")
APPEND MENU ITEM(subMenu;"SubTest2") // サブメニューに項目を追加する

APPEND MENU ITEM(newMenu;"Test2";subMenu) // メニューにサブメニューを追加

  // この時点でサブメニューがメニューに追加されています。あとでこのサブメニューを使用しなのであれば、この時点で削除を行うことができます。
  // 削除を行っても即座にsubMenuが消去されるわけではありません。それはサブメニューがまだnewMenuにより使用されているからです。subMenuはnewMenuが削除されたときに、削除されます。
  // ここでサブメニューを削除することでメモリを節約できます。
RELEASE MENU(subMenu)

$result1:=Dynamic pop up menu(newMenu) // メニューを使用する
copyMenu:=Create menu(newMenu) // newMenuをコピーして新しいメニューを作成する (subMenuもコピーされます)
RELEASE MENU(newMenu) // newMenuはもう必要ありません

$result2:=Dynamic pop up menu(copyMenu)
RELEASE MENU(copyMenu)
  // copyMenuのサブメニューについて考慮する必要はありません。サブメニューはCreate menuを使用して作成されたものではないからです。
  // 従うべきルールは、各Create menuに対応するRELEASE MENUを呼び出すことです。
```

SET MENU BAR

SET MENU BAR (menuBar {; process}{; *})

引数	型	説明
menuBar	倍長整数, 文字, MenuRef	⇒ メニューバー番号または名前 または メニュー参照
process	倍長整数	⇒ プロセス参照番号
*	演算子	⇒ メニューバーの状態を保存

説明

SET MENU BAR はカレントプロセスのみのメニューバーをmenuBarで指定したメニューバーで置き換えます。menuBar 引数には、新しいメニューバーの番号または名前を渡します。またメニューID (MenuRef 型, 16文字の文字列) を渡すこともできます。参照を使用する場合、メニューをメニューバーとして、あるいはその逆として使用できます (の節を参照)。

Note: メニューバー名には31文字までを指定でき、ユニークでなければなりません。

オプションのprocess 引数を使用すると、指定したプロセスのメニューバーをmenuBarに変更します。

Note: menuBarにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

オプションの引数 * を使用すると、メニューバーの状態を保存できます。この引数が省略された場合、SET MENU BARが実行されたとき、コマンドはメニューバーを再初期化します。

例えばSET MENU BAR(1)を実行したとします。次にDISABLE MENU ITEMコマンドを使い、いくつかのメニューを使用不可にします。

SET MENU BAR**(1)**を2度目に実行すると、その実行が同じプロセスからでも別のプロセスからでも、メニューはすべて、元の使用可の状態に戻ります。

SET MENU BAR**(1;*)**を実行すると、メニューバーは前と同じ状態を保っており、使用不可にしたメニューは使用不可のままです。

Note: menuBarにMenuRefを渡した場合、* 引数は意味を持たず、無視されます。

ユーザがアプリケーションモードに移ると、最初のメニューバー (メニューバー#1) が表示されます。データベースを開く際にや個々のユーザ用のStartupメソッドで目的のメニューバーを指定して、メニューバーを変更することができます。

例題 1

以下の例は、カレントメニューバーをメニューバー#3に変更し、メニューの状態を元に戻します:

```
SET MENU BAR(3)
```

例題 2

以下の例題は、カレントメニューバーを"FormMenuBar1"という名前のメニューバーに変更し、メニューコマンドの状態を保存します。事前に選択不可に設定されたメニューコマンドは、選択不可のまま表示されます。

```
SET MENU BAR("FormMenuBar1";*)
```

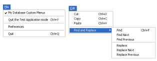
例題 3

以下の例は、レコードの変更中フォームのメニューバーをメニューバー#3に変更します。レコードの変更が終了すると、メニューの状態を保存してメニューバーをメニューバー#2に戻します:

```
SET MENU BAR(3)
ALL RECORDS([Customers])
MODIFY SELECTION([Customers])
SET MENU BAR(2;*)
```

例題 4

この包括的な例題では、以下のファイルメニューや編集メニューをプログラムで作成します:



```
  //File メニューを作成するメソッド
 C_TEXT(FileMenu) //FileMenu にはFIleメニュー参照が含まれます
 FileMenu:=Create menu
 INSERT MENU ITEM(FileMenu;-1;"My Database "+Get indexed string(131;29))
 SET MENU ITEM MARK(FileMenu;1;Char(18))
 INSERT MENU ITEM(FileMenu;-1;"(-")
 INSERT MENU ITEM(FileMenu;-1;"Quit the Test Application mode/Y")
 SET MENU ITEM PROPERTY(FileMenu;3;Associated Standard Action;Return to Design mode)
 INSERT MENU ITEM(FileMenu;-1;"(-")
 INSERT MENU ITEM(FileMenu;-1;Get indexed string(131;26))
 SET MENU ITEM PROPERTY(FileMenu;6;Associated Standard Action;Database Settings Action) //Settings
 INSERT MENU ITEM(FileMenu;-1;"(-")
 INSERT MENU ITEM(FileMenu;-1;Get indexed string(131;30))
 SET MENU ITEM PROPERTY(FileMenu;7;Associated Standard Action;Quit Action) //Quit
 SET MENU ITEM SHORTCUT(FileMenu;7;Character code("Q"))

  //Find and Replace メニューを作成するメソッド
 C_TEXT(FindAndReplaceMenu) //FindAndReplaceMenu にはFind and Replaceメニュー参照が含まれます
 FindAndReplaceMenu:=Create menu
 APPEND MENU ITEM(FindAndReplaceMenu;"Find;Find Next;Find Previous;(-;Replace;Replace Next;Replace Previous")
 SET MENU ITEM SHORTCUT(FindAndReplaceMenu;1;Character code("F"))
 SET MENU ITEM SHORTCUT(FindAndReplaceMenu;5;Character code("R"))
 SET MENU ITEM METHOD(FindAndReplaceMenu;1;"MyFindMethod")

  //Edit メニューを作成するメソッド
 C_TEXT(EditMenu) //EditMenu には Edit メニュー参照が含まれます
 EditMenu:=Create menu
 APPEND MENU ITEM(EditMenu;"Cut;Copy;Paste")
 SET MENU ITEM SHORTCUT(EditMenu;1;Character code("X"))
 SET MENU ITEM PROPERTY(EditMenu;1;Associated Standard Action;Cut Action)
 SET MENU ITEM SHORTCUT(EditMenu;2;Character code("C"))
 SET MENU ITEM PROPERTY(EditMenu;2;Associated Standard Action;Copy Action)
 SET MENU ITEM SHORTCUT(EditMenu;3;Character code("V"))
 SET MENU ITEM PROPERTY(EditMenu;3;Associated Standard Action;Paste Action)
 INSERT MENU ITEM(EditMenu;-1;"(-")
 INSERT MENU ITEM(EditMenu;-1;"Find and Replace";FindAndReplaceMenu) //サブメニューを持つアイテム

 main_Bar:=Create menu //他のメニューを統合してメニューバーを作成する
 INSERT MENU ITEM(main_Bar;-1;Get indexed string(79;1);FileMenu)
 APPEND MENU ITEM(main_Bar;"Edit";EditMenu)

 SET MENU BAR(main_Bar<span style="color: #000000;"><span style="font-family: arial,helvetica,clean,sans-
serif;">)</span></span>
```

SET MENU ITEM

SET MENU ITEM (menu ; menuItem ; itemText {; process}{; *})

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー番号またはメニュー参照
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
itemText	文字	→	メニュー項目の新しいテキスト
process	倍長整数	→	プロセス参照番号
*	演算子	→	指定時: メタ文字を標準文字として扱う

説明

SET MENU ITEM コマンドは、menu引数にメニュー番号またはメニュー参照で指定したメニュー中、menuItem引数にメニュー項目番号で指定したメニュー項目のテキストを、itemTextに渡したテキストに変更します。menuItemに-1を渡すと、menuに最後に追加された項目を指定したことになります。

* 引数を渡さない場合、itemTextに含められた特別文字 (;や !) はメタ文字として扱われます。例えばメニュー項目を区切り線に設定する場合、"-"文字をitemTextに渡します。
* 引数を渡すと特別文字は標準の文字として扱われます。これらの文字についてはAPPEND MENU ITEM コマンドの説明を参照してください。

process引数を省略すると、SET MENU ITEMコマンドはカレントプロセスのメニューバーに適用されます。そうでなければ、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡した場合、process引数は意味を持たず、無視されます。

SET MENU ITEM ICON

SET MENU ITEM ICON (menu ; menuItem ; iconRef {; process})

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー参照またはメニュー番号
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
iconRef	テキスト, 倍長整数	→	メニュー項目に関連付けられた ピクチャの番号または名前
process	倍長整数	→	プロセス番号

説明

SET MENU ITEM ICON コマンドは、menuとmenuItem引数で指定されたメニュー項目に関連付けるアイコンを変更するために使用します。menuItemに-1を渡してmenuに最後に追加された項目を指定することができます。

menuにはメニュー参照 (MenuRef) またはメニュー番号を渡します。メニュー参照を渡す場合、process 引数は必要なく、渡されても無視されます。メニュー番号を渡す場合、コマンドはカレントプロセスのメインメニューバー中の対応するメニューを考慮します。他のプロセスを指定したい場合、オプションのprocess 引数にその番号を渡します。

メニュー項目に関連付けられたアイコンは、アプリケーションのツールバーに追加されます。ピクチャは20 x 20ピクセルのフレームに表示されます。

iconRefには、アイコンとして使用するピクチャを渡します。ライブラリのピクチャまたはピクチャ参照を使用できます。

- ライブラリピクチャ: ピクチャの名前または番号を渡します。名前が重複することがあるのに対しピクチャ番号はユニークであるため、名前よりも番号を使用することが一般に推奨されます。
- ピクチャ参照: ピクチャはデータベースの**Resources**フォルダに格納されていて、iconRef には"file:{pathname}fileName"のシンタックスを使用しなければなりません。**Resources** フォルダに関する詳細はの節を参照してください。

例題

データベースのResourcesフォルダにあるピクチャを使用する:

```
SET MENU ITEM ICON($MenuRef;2;"File:English.lproj/spot.png")
```

SET MENU ITEM MARK

SET MENU ITEM MARK (menu ; menuItem ; mark {; process})

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー番号またはメニュー参照
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
mark	文字	→	新しいメニュー項目マーク
process	倍長整数	→	プロセス参照番号

説明

SET MENU ITEM MARK コマンドは、menu引数にメニュー番号またはメニュー参照で指定したメニュー中、menuItem引数にメニュー項目番号で指定したメニュー項目のチェックマークを、markに渡した最初の文字に変更します。

process引数を省略すると、SET MENU ITEM MARKコマンドはカレントプロセスのメニューバーに適用されます。そうでなければ、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

空の文字列を渡すと、メニュー項目からチェックマークが取り除かれます。そうでなければ:

- Macintoshでは、文字列の最初の文字がメニュー項目のマークになります。通常はMacintoshメニューのマークであるChar(18)を渡します。
- Windowsでは標準のチェックマークが割り当てられます。

例題

Get menu item mark コマンドの例題を参照

SET MENU ITEM METHOD

SET MENU ITEM METHOD (menu ; menuItem ; methodName {; process})

引数	型		説明
menu	倍長整数, MenuRef	⇒	メニュー参照またはメニュー番号
menuItem	倍長整数	⇒	メニュー項目番号 または -1: 最後に追加された項目
methodName	文字	⇒	メソッド名
process	倍長整数	⇒	プロセス番号

説明

SET MENU ITEM METHOD コマンドは、menuとmenuItem引数で指定されたメニュー項目に関連付ける4Dプロジェクトメソッドを変更するために使用します。
menuItemに-1を渡してmenuに最後に追加された項目を指定することもできます。

menuにはメニュー参照 (MenuRef) またはメニュー番号を渡します。メニュー参照を渡す場合、コマンドはすべてのプロセスのすべてのインスタンスに適用されます。この場合process引数は渡されても無視されます。メニュー番号を渡す場合、コマンドはカレントプロセスのメインメニューバー中の対応するメニューに適用されます。他のプロセスを指定したい場合、オプションのprocess 引数にその番号を渡します。

methodには、4Dメソッド名を文字列で渡します。

Note: メニューが階層サブメニューのタイトルに対応する場合、メニューが選択されてもメソッドは呼び出されません。

例題

SET MENU BAR コマンドの例題参照

SET MENU ITEM PARAMETER

SET MENU ITEM PARAMETER (menu ; menuItem ; param)

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー参照またはメニュー番号
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
param	文字	→	パラメタとして割り当てる文字列

説明

SET MENU ITEM PARAMETERコマンドは、menuとmenuItem引数で指定されたメニュー項目に、カスタム文字列を設定するために使用します。

このパラメタは主にDynamic pop up menuコマンドで使用されます。

例題

以下のコードでは開かれたウィンドウの名前で構成されるメニューを作成し、選択されたウィンドウの番号を取得できるようにします:

```
WINDOW LIST($alWindow)
$tMenuRef:=Create menu
For($i;1;Size of array($alWindow))
   APPEND MENU ITEM($tMenuRef;Get window title($alWindow{$i})) // メニュー項目のタイトル
   SET MENU ITEM PARAMETER($tMenuRef;-1;String($alWindow{$i})) // メニュー項目から返される値
End for
$tWindowRef:=Dynamic pop up menu($tMenuRef)
RELEASE MENU($tMenuRef)
```

SET MENU ITEM PROPERTY

SET MENU ITEM PROPERTY (menu ; menuItem ; property ; value {; process})

引数	型	説明
menu	倍長整数, MenuRef	→メニュー参照またはメニュー番号
menuItem	倍長整数	→メニュー項目番号 または -1: 最後に追加された項目
property	文字	→プロパティタイプ
value	式	→プロパティ値
process	倍長整数	→プロセス番号

説明

SET MENU ITEM PROPERTYコマンドは、menuと menuItem引数で指定されたメニュー項目に、propertyのvalueを設定するために使用します。
menuItemに-1を渡してmenuに最後に追加された項目を指定することもできます。

menuにはメニュー参照 (MenuRef) またはメニュー番号を渡します。メニュー参照を渡す場合、コマンドはすべてのプロセスのすべてのインスタンスに適用されます。この場合process 引数は渡されても無視されます。メニュー番号を渡す場合、コマンドはカレントプロセスのメインメニューバー中の対応するメニューに適用されます。他のプロセスを指定したい場合、オプションのprocess 引数にその番号を渡します。

property 引数には値を変更するプロパティを、そして新しい値をvalueに渡します。property 引数には、Menu Item Propertiesテーマの定数またはカスタム値を指定できます:

標準のプロパティ: Menu Item Propertiesテーマの定数および取りうる値を以下で説明します。Associated Standard Action プロパティの場合、value 引数にはValue for Associated Standard Actionテーマの定数を渡すことができます:

定数	型	値	コメント
Access Privileges	文字列	4D_access_group	コマンドにアクセスグループを設定するために使用します。 0 = All Groups >0 = Group ID
Associated Standard Action	文字列	4D_standard_action	メニュー項目に標準アクションを割り当てるために使用します。 "Value for Associated Standard Action" テーマの定数参照。
Start a New Process	文字列	4D_start_new_process	"新規プロセス開始オプションを有効にします。 0 = No, 1 = Yes

以下はValue for Associated Standard Actionテーマの定数です:

定数	型	値
Accept Action	倍長整数	2
Add subrecord Action	倍長整数	14
Cancel Action	倍長整数	1
Clear Action	倍長整数	21
Copy Action	倍長整数	19
Cut Action	倍長整数	18
Database Settings Action	倍長整数	32
Delete record Action	倍長整数	7
Delete subrecord Action	倍長整数	13
Edit subrecord Action	倍長整数	12
First page Action	倍長整数	10
First record Action	倍長整数	5
Last page Action	倍長整数	11
Last record Action	倍長整数	6
MSC Action	倍長整数	36
Next page Action	倍長整数	8
Next record Action	倍長整数	3
No Action	倍長整数	0
Paste Action	倍長整数	20
Previous page action	倍長整数	9
Previous record Action	倍長整数	4
Quit Action	倍長整数	27
Redo Action	倍長整数	31
Return to Design mode	倍長整数	35
Select all Action	倍長整数	22
Show Clipboard Action	倍長整数	23
Test Application Action	倍長整数	26
Undo Action	倍長整数	17

標準のメニュー項目プロパティに関する詳細は、Design Referenceマニュアルの"カスタムメニューの作成"の章を参照してください。

カスタムプロパティ

propertyにカスタムテキストを渡して、valueにテキスト、数値、ブール型を割り当てることができます。このvalueは項目に格納され、GET MENU ITEM PROPERTY コマンドを使用して取り出すことができます。property 引数には任意のカスタム文字列を使用できます。ただし4Dが使用するタイトルを使用しないように注意してください (慣例で、4Dが設定するプロパティは"4D_"で始まります)。

注: メニューが階層サブメニューのタイトルに対応する場合、メニューが選択されても標準アクションは呼び出されません。

SET MENU ITEM SHORTCUT

SET MENU ITEM SHORTCUT (menu ; menuItem ; itemKey ; modifiers {; process})

引数	型		説明
menu	倍長整数, MenuRef	⇒	メニュー番号またはメニュー参照
menuItem	倍長整数	⇒	メニュー項目番号 または -1: 最後に追加された項目
itemKey	文字, 倍長整数	⇒	キーボードショートカットの文字またはキーボードショートカットの文字コード (古いシンタックス)
modifiers	倍長整数	⇒	ショートカットに割り当てられたモディファイア (キーコードが渡された時は無視)
process	倍長整数	⇒	プロセス参照番号

説明

SET MENU ITEM SHORTCUT コマンドは、menuとmenuItem引数で指定されたメニュー項目に割り当てられるショートカットキーを、itemKeyに渡された文字コードまたは文字で置き換えます。menuItemに-1を渡してmenuに最後に追加された項目を指定することもできます。新しいキーボードショートカットを設定する際に、このキーは**Ctrl** (Windows) または **Command** (Macintosh) キーと組み合わされます。

itemKey引数に文字を渡して、ショートカットキーを指定できます。例えば "U" は **Ctrl+U** (Windows) または **Command+U** (Mac OS) ショートカットを指定します。modifiers 引数を渡してショートカットに追加のモディファイアを指定することができます。この方法で **Ctrl+Alt+Shift+Z** (Windows) や **Cmd+Option+Shift+Z** (Mac OS) のようなショートカットを定義できます。

以下の値をmodifiersに渡すことができます:

- 256: **Command** (Mac OS) または**Ctrl** (Windows) キー
- 512: **Shift** キー
- 2048: **Option** (Mac OS) または **Alt** (Windows) キー
- キーの組み合わせを割り当てるにはこれらの値を加算します。

注: Events (Modifiers)テーマのCommand key mask、Shift key mask、Option key mask定数を使用して値を渡すこともできます。

Ctrl (Windows) と **Command** (Mac OS) キーだけがモディファイアキーである場合、値256または対応する定数をmodifiers引数に渡す必要があります。これらのキーに加え複数のモディファイアキーを使用する場合、**Ctrl** (Windows) と **Command** (Mac OS) キーは自動でキーボードショートカットに追加されます。そのため値256をこの引数に追加する必要はありません。

注:互換性のため、コマンドはitemKey引数に文字コードを受け入れます (過去のシンタックス)。この場合modifiers 引数は無視され、省略できます。ショートカットには**Ctrl** (Windows) または **Command** (Mac OS) キーだけが割り当てられます。

process引数を省略すると、SET MENU ITEM SHORTCUTコマンドはカレントプロセスのメニューバーに適用されます。そうでなければ、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

注: menuにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

itemKeyに0を渡すと、メニュー項目からショートカットキーが取り除かれます。

例題 1

"下線"メニュー項目に Ctrl+Shift+U (Windows) と Cmd+Shift+U (Mac OS) ショートカットを定義します:

```
SET MENU ITEM(menuRef;1;"Underline")
SET MENU ITEM SHORTCUT(menuRef;1;"U";Shift key mask)
```

例題 2

Ctrl+R (Windows) や Cmd+R (Mac OS) ショートカットを"再起動"メニュー項目に割り当てます:

```
INSERT MENU ITEM(FileMenu;-1;"Restart")
SET MENU ITEM SHORTCUT(FileMenu;-1;"R";Command key mask)
```

SET MENU ITEM STYLE

SET MENU ITEM STYLE (menu ; menuItem ; itemStyle {; process})

引数	型		説明
menu	倍長整数, MenuRef	→	メニュー番号またはメニュー参照
menuItem	倍長整数	→	メニュー項目番号 または -1: 最後に追加された項目
itemStyle	倍長整数	→	新しいメニュー項目スタイル
process	倍長整数	→	プロセス参照番号

説明

SET MENU ITEM STYLE コマンドは、menu引数にメニュー番号またはメニュー参照で指定したメニュー中、menuItem引数にメニュー項目番号で指定したメニュー項目のフォントスタイルを、itemStyleに変更します。

process引数を省略すると、SET MENU ITEM STYLEコマンドはカレントプロセスのメニューバーに適用されます。そうでなければ、processに渡されたプロセス参照番号のプロセスのメニューに適用されます。

Note: menuにMenuRefを渡した場合、process 引数は意味を持たず、無視されます。

itemStyle 引数には項目のフォントスタイルを指定します。""テーマの以下の定義済み定数と1つあるいは加算して渡します:

定数	型	値
Bold	倍長整数	1
Italic	倍長整数	2
Plain	倍長整数	0
Underline	倍長整数	4

ユーザ＆グループ

- BLOB TO USERS
- CHANGE CURRENT USER
- CHANGE LICENSES
- CHANGE PASSWORD
- Current user
- DELETE USER
- EDIT ACCESS
- Get default user
- GET GROUP LIST
- GET GROUP PROPERTIES
- Get plugin access
- GET USER LIST
- GET USER PROPERTIES
- Is license available
- Is user deleted
- Set group properties
- SET PLUGIN ACCESS
- Set user properties
- User in group
- USERS TO BLOB
- Validate password Updated 14.0

BLOB TO USERS

BLOB TO USERS (users)

引数	型		説明
users	BLOB	→	データベース管理者によって作成され保存された データベースユーザアカウントを含む (暗号化された) BLOB

説明

BLOB TO USERS コマンドは、BLOB users に存在するユーザアカウントをデータベースに追加します。BLOB users は暗号化されています。そして、USERS TO BLOB コマンドを用いて作成されていなければなりません。

データベース管理者および設計者のみが、このコマンドを実行できます。他のユーザが実行しようとすると、コマンドは何も行わず、権限エラー (-9949) が生成されます。

ユーザアカウントを追加する際、以下のルールが適用されます。

- ユーザIDは参照として使用されます。ユーザは、自身のユーザIDに応じて順番に処理されます。
- 番号が既にストラクチャファイルに存在する場合、必要に応じて、既存のアカウント情報がBLOBに存在する情報に基づいて更新されます。
- 番号がストラクチャファイルに存在しない場合、BLOBに存在する情報に基づいてユーザが作成されます。
- 番号がストラクチャファイル内で削除されたユーザアカウントと同じである場合、アカウントはBLOBに存在する情報に基づいて更新されます。
- BLOBに存在する情報内でユーザ情報が削除されていると、そのアカウントはストラクチャファイルから削除されます。
- 更新されたユーザは、BLOBに存在する情報に基づいてグループにリンクされます。
- グループが存在しない場合、そのグループは作成されます。

コマンドが正確に実行されるとシステム変数OKに1が代入されます。その他の場合は0が代入されます。

互換性に関するメモ: バージョン2004より前の4Dで**グループを保存...** メニューコマンドを用いて作成されたユーザとグループのファイル (.4UG extension) を以下のコードを用いて4Dのバージョン2004以降でロードすることが可能です。

```
DOCUMENT TO BLOB(mydoc;blob)
BLOB TO USERS(blob)
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKには1が、そうでなければ0が設定されます。

CHANGE CURRENT USER

CHANGE CURRENT USER {(user ; password)}

引数	型	説明
user	文字, 倍長整数	⇒名前またはユニークなユーザID
password	文字	⇒パスワード (暗号化されていない)

説明

CHANGE CURRENT USER コマンドを使用すると、データベースを終了させずに、現在のユーザを変更できます。ユーザはデータベース接続ダイアログボックス (引数なしでコマンドが呼び出された場合) を使用して、またはコマンドから直接、ログインユーザを変更することが可能です。ユーザがログインを変更すると、選択されたユーザが保有する権限を優先するため、以前のアクセス権はすべて放棄されます。

引数なしでコマンドが実行されると、データベース接続ダイアログボックスが表示されます。データベースへ入るには、ユーザは必ず有効な名前とパスワードを入力または選択しなければなりません。接続ダイアログボックスの内容は、データベース環境設定の **アプリケーション/アクセス**ページに依存します。

また、使用するアカウントをプログラミングで指定するには、2つのオプション引数 user と password を渡します。
引数 user には、使用するアカウントの名前またはユニークなユーザID (userRef) を渡します。GET USER LIST コマンドを用いて、ユーザ名とIDを取得できます。

ユーザID	ユーザ説明
1	Designer
2	Administrator
3 ~ 15000	Designerによって作成されたユーザ。(ユーザ番号3は、Designerによって最初に作成されたユーザ、ユーザ番号4は2番目に作成されたユーザなど)
-11 ~ -15010	Administratorによって作成されたユーザ。(ユーザ番号-11は、Administratorによって最初に作成されたユーザ、ユーザ番号-12は、2番目に作成されたユーザなど)

ユーザアカウントが存在しない場合や削除された場合、エラーコード -9979が返されます。ON ERR CALL コマンドを用いてインストールされたエラー処理メソッドでこのエラーを検知することができます。その他ではIs user deletedコマンドを呼び出してユーザアカウントをテストし、その後このコマンドを呼び出す方法があります。

暗号化されていないユーザアカウントのパスワードを引数 password に渡します。パスワードがユーザと一致しない場合、コマンドはエラーメッセージ -9978を返し、何も行いません。

フラッディング (ブルートフォース攻撃) 、つまり、複数のユーザ名とパスワードの組み合わせによる試みを防ぐために、コマンドは遅れて実行されます。その結果、このコマンドを4回呼び出した後は、10秒経った後にコマンドは実行されます。この遅れは、ワークステーション全体を通して発生します。

カスタムアクセス管理ダイアログボックスの提供

CHANGE CURRENT USER コマンドを使用して、名前とパスワード (入力と有効期限のルール付きの) を入力するためのカスタムダイアログボックスを設定することができます。このダイアログボックスには、4Dのアクセスコントロールシステムと同じメリットがあります。

次のように機能します。

1. "デフォルトユーザ" モードでは、ダイアログボックスは表示されず、直接データベースへ入ります。
2. は、ユーザ名とパスワードを入力するためのカスタムダイアログボックスを表示します。実行可能な処理はすべてダイアログに表示されています。
   - 4Dの標準アクセスダイアログボックスと同様に、GET USER LIST コマンドを使用してデータベースのユーザリストを表示できます。
   - 入力された文字の有効性をチェックするための様々な規制 (最低文字数、ユニークなど) を、パスワード入力フィールドにつけられます。
   - 入力中のパスワードの文字を画面上でマスクするには、FILTER KEYSTROKE コマンドと特別な%password フォントを使用します。
   - 有効期限に関する規則は、ダイアログボックスを受け入れる時に適用できます。期限切れ、初期の接続への変更、数回に及ぶ不正入力後のアカウントロック、既に使用されたパスワードのメモリーなど。
3. 入力が有効になると、ユーザのアカウント権限でデータベースを開くために、必須情報 (ユーザ名とパスワード) が CHANGE CURRENT USER コマンドに渡されます。

例題

以下の例題を使用して、接続ダイアログボックスを表示します。

```
CHANGE CURRENT USER
```

CHANGE LICENSES

CHANGE LICENSES
このコマンドは引数を必要としません

説明

CHANGE LICENSES コマンドは、4Dライセンス更新ダイアログボックスを表示します。このダイアログボックスでは、データベースとそのプラグインを同時に使用することができるユーザ数を増やすために、4Dサーバを用いてプラグインやWebサーバをアクティブにしたり、Expansion番号を付けたりすることができます。

Note: 4Dや4D Serverでは、**ヘルプ**メニューにある**ライセンス更新...** コマンドを選択することによって、このダイアログボックスを表示できます。

CHANGE LICENSES コマンドを使用して、4Dライセンスダイアログボックスをアプリケーションモードから呼び出します。

CHANGE LICENSES は、顧客へ配布されるコンパイルされた4Dアプリケーションにライセンスを許可する便利な方法です。4DデベロッパまたはISマネージャは配布するアプリケーションでこのコマンドを使用し、ユーザにライセンスを入力させるようにできます。

このダイアログボックスに関する詳細は、4Dインストールガイドを参照してください。

例題

カスタム設定または環境設定ダイアログボックス内に、以下のメソッドを持つボタンを設定します。

```
 ` bライセンスボタンオブジェクトメソッド
CHANGE LICENSES
```

CHANGE PASSWORD

CHANGE PASSWORD (password)

引数	型		説明
password	文字	→	新しいパスワード

説明

CHANGE PASSWORD コマンドを使用して、カレントユーザのパスワードを変更できます。このコマンドは現在のパスワードを、引数 password に渡した新しいパスワードに置き換えます。

警告: パスワードでは大文字小文字が区別されます。

例題

以下の例題を使用して、ユーザがパスワードの変更を行います。

```
CHANGE CURRENT USER `ユーザにパスワードダイアログを表示する
If(OK=1)
   $pw1:=Request("Enter new password for "+Current user)
 `パスワードは少なくとも5文字以上にする
   If(((OK=1)&($pw1#""))&(Length($pw1)>5))
 `パスワードが正確に入力されたことを確認する
      $pw2:=Request("Enter password again")
      If((OK=1)&($pw1=$pw2))
         CHANGE PASSWORD($pw2) `パスワードの変更
      End if
   End if
End if
```

Current user

Current user -> 戻り値

引数	型		説明
戻り値	文字	←	カレントユーザのユーザ名

説明

Current user コマンドは、カレントユーザのユーザ名を返します。

例題

User in group コマンドの例題を参照してください。

DELETE USER

DELETE USER (userID)

引数	型		説明
userID	倍長整数	→	削除するユーザのID番号

説明

DELETE USER コマンドは、引数 userID に渡したユニークなユーザID番号を持つユーザを削除します。この場合、必ずGET USER LIST コマンドによって返された有効なユーザID番号を渡さなければなりません。

ユーザアカウントが存在しない場合や既に削除されている場合は、エラーコード -9979が生成されます。ON ERR CALLコマンドを用いてインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

DesignerとAdministratorのみがユーザを削除できます。AdministratorはDesignerによって作成されたユーザを削除するこはできません。

削除されたユーザー名は、EDIT ACCESSコマンドやデザインモードで表示されるユーザーエディターに表示されなくなります。削除されたユーザーの番号は、新しいユーザーアカウントが作成される際に再割り当てされることがあることに留意してください。

エラー管理

コマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによって既にアクセスされている場合は、アクセス権エラーが生成されます。ON ERR CALL コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

EDIT ACCESS

EDIT ACCESS
このコマンドは引数を必要としません

説明

EDIT ACCESS コマンドを使用して、ユーザにパスワードシステムの編集環境を提供します。このコマンドが実行されると、ユーザとユーザグループページのみから成るツールボックスウィンドウが表示されます。

Note: このコマンドはモーダルウィンドウを開きます。従って、他のモーダルウィンドウからこのコマンドを呼び出すことはできません。呼び出した場合、コマンドは何も行いません。

Designer、Administrator、およびグループオーナーはグループを編集することができます。DesignerとAdministratorはすべてのグループを編集できますが、グループオーナーは所有するグループのみ編集できます。ユーザをグループへ追加したり、削除したりすることが可能です。グループが指定されていない場合、コマンドは機能しません。

DesignerとAdministratorは新規ユーザの追加、およびグループへの割り当てができます。

例題

以下の例は、ユーザに対してユーザグループ管理ウィンドウを表示します。

```
EDIT ACCESS
```

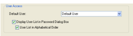
Get default user

Get default user -> 戻り値

引数	型		説明
戻り値	倍長整数	⮐	ユニークなユーザID番号

説明

Get default userコマンドは、データベースの環境設定ダイアログボックスにおいて "デフォルトユーザ" として設定されたユーザのユニークなユーザIDを返します。



User Access

Default User: Default User

☑ Display User List in Password Dialog Box

☑ User List in Alphabetical Order

ユーザIDとして以下の番号を使用します。

ID	ユーザ説明
1	Designer
2	Administrator
3 to 15000	Designerによって作成されたユーザ (ユーザ番号3は、Designerによって最初に作成されたユーザ、ユーザ番号4は、2番目に作成されたユーザなど)
-11 to -15010	Administratorによって作成されたユーザ (ユーザ番号-11は、Administratorによって最初に作成されたユーザ、ユーザ番号-12は、2番目に作成されたユーザなど)

デフォルトユーザが設定されていない場合、コマンドは0を返します。

GET GROUP LIST

GET GROUP LIST (groupNames ; groupNumbers)

引数	型		説明
groupNames	文字配列	←	パスワードエディタウィンドウに 表示されるグループの名前
groupNumbers	倍長整数配列	←	対応するユニークなグループID番号

説明

GET GROUP LIST コマンドは、パスワードエディタウィンドウに表示されるグループの名前とユニークなID番号を配列 groupNames と groupNumbers に割り当てます。

配列 groupNames と同期される配列 groupNumbers には、対応するユニークなグループID番号が代入されます。これらの番号は以下の値および範囲を持っています。

グループID番号	グループ説明
15001 to 32767	Designerまたは関連するグループオーナーによって作成されたグループ。(グループ番号15001は、Designerによって最初に作成されたグループ、グループ番号15002は、2番目に作成されたグループなど)
-15001 to -32768	Administratorまたは関連するグループオーナーによって作成されたグループ。(グループ番号-15001はAdministratorによって最初に作成されたグループ、グループ番号-15002は、2番目に作成されたグループなど)

エラー管理

コマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによって既にアクセスされている場合は、アクセス権エラーが生成されます。ON ERR CALL コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

GET GROUP PROPERTIES

GET GROUP PROPERTIES (groupID ; name ; owner {; members})

引数	型		説明
groupID	倍長整数	→	ユニークなグループID番号
name	文字	←	グループの名前
owner	倍長整数	←	グループオーナーのユーザID番号
members	倍長整数配列	←	グループメンバー

説明

GET GROUP PROPERTIES コマンドは、引数 groupID に渡したユニークなグループID番号を持つグループのプロパティを返します。GET GROUP LIST コマンドによって返された有効なグループID番号を必ず渡さなければなりません。グループID番号は以下の値および範囲を持っています。

グループID番号	グループ説明
15001 to 32767	Designerまたは関連するグループオーナーによって作成されたグループ。(グループ番号15001はDesignerによって最初に作成されたグループ、グループ番号15002は、2番目に作成されたグループなど)
-15001 to -32768	Administratorまたは関連するグループオーナーによって作成されたグループ。(グループ番号-15001は、Administratorによって最初に作成されたグループ、グループ番号-15002は、2番目に作成されたグループなど)

有効なグループID番号を渡されない場合、GET GROUP PROPERTIES コマンドは空の引数を返します。

呼び出し後、引数 name と owner にグループの名前とオーナーが返されます。

オプション引数 members を渡すと、グループに属するユーザとグループのユニークなID番号が返されます。メンバーID番号は以下の値および範囲を持っています。

メンバーID番号	メンバー説明
1	Designerユーザ
2	Administratorユーザ
3 to 15000	データベースのDesignerによって作成されたユーザ。(ユーザ番号3は、Designerによって最初に作成されたユーザ、ユーザ番号4は、2番目に作成されたユーザなど)
-11 to -15000	データベースのAdministratorによって作成されたユーザ。(ユーザ番号-11は、Administratorによって最初に作成されたユーザ、ユーザ番号-12は、2番目に作成されたユーザなど)
15001 to 32767	Designerまたは関連するグループオーナーによって作成されたグループ。(グループ番号15001は、Designerによって最初に作成されたグループ、グループ番号15002は、2番目に作成されたグループなど)
-15001 to -32768	Administratorまたは関連するグループオーナーによって作成されたグループ。(グループ番号-15001は、Administratorによって最初に作成されたグループ、グループ番号-15002は、2番目に作成されたグループなど)

エラー管理

コマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによって既にアクセスされている場合は、アクセス権エラーが生成されます。ON ERR CALL コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

Get plugin access

Get plugin access (plugIn) -> 戻り値

引数	型		説明
plugIn	倍長整数	→	プラグイン番号
戻り値	文字	←	プラグインに割り当てられたグループ名

説明

Get plugin accessコマンドは引数 plugIn に渡した番号を持つプラグインの使用を許可されているユーザグループの名前を返します。プラグインに割り当てられているグループが存在しない場合、コマンドは空の文字列 ("") を返します。

プラグインの番号に割り当てられているユーザのグループを調べたい場合、そのプラグイン番号を引数 plugIn に渡します。プラグインのライセンスには4DクライアントWebとSOAPライセンスがあります。"" テーマにある以下の定数の1つを渡します。

定数	型	値
4D Client SOAP License	倍長整数	808465465
4D Client Web License	倍長整数	808465209
4D Draw License	倍長整数	808464694
4D for ADO License	倍長整数	808465714
4D for MySQL License	倍長整数	808465712
4D for OCI License	倍長整数	808465208
4D for PostgreSQL License	倍長整数	808465713
4D for Sybase License	倍長整数	808465715
4D ODBC Pro License	倍長整数	808464946
4D View License	倍長整数	808465207
4D Write License	倍長整数	808464697

GET USER LIST

GET USER LIST (userNames ; userNumbers)

引数	型		説明
userNames	文字配列	⇐	パスワードエディタウィンドウに 表示されるユーザの名前
userNumbers	倍長整数配列	⇐	対応するユニークなユーザID番号

説明

GET USER LIST コマンドは、パスワードウィンドウに表示されるユーザの名前とユニークなID番号を配列 userNames と userNumbers に割り当てます。

配列 userNames には、パスワードウィンドウで表示されたユーザ名が代入されます。これには、無効となったアカウントを持つユーザも含まれます (パスワードウィンドウではユーザ名を緑色で表示されます) 。

Note: 削除されたユーザを見つけるには、Is user deleted コマンドを使用してください。

配列 userNames と同期される配列 userNumbers には、対応するユニークなユーザID番号が代入されます。これらの番号は以下の値および範囲を持っています。

ユーザID番号	ユーザ説明
1	設計者
2	管理者
3 to 15000	データベースの設計者によって作成されたユーザ (ユーザ番号3は、設計者によって最初に作成されたユーザ、ユーザ番号4は、2番目に作成されたユーザなど)
-11 to -15000	データベースの管理者によって作成されたユーザ (ユーザ番号-11は、管理者によって最初に作成されたユーザ、ユーザ番号-12は、2番目に作成されたユーザなど)

エラー管理

コマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによって既にアクセスされている場合は、アクセス権エラーが生成されます。ON ERR CALL コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

GET USER PROPERTIES

GET USER PROPERTIES (userID ; name ; startup ; password ; nbLogin ; lastLogin {; memberships {; groupOwner}})

引数	型		説明
userID	倍長整数	→	ユニークなユーザID番号
name	文字	←	ユーザの名前
startup	文字	←	スタートアップメソッドの名前
password	文字	←	常に空の文字列
nbLogin	倍長整数	←	データベースにログインした数
lastLogin	日付	←	データベースに最後にログインした日付
memberships	倍長整数配列	←	ユーザが属するグループのID番号
groupOwner	倍長整数	←	ユーザのグループオーナーのID番号

説明

GET USER PROPERTIES コマンドは、引数 userID に渡したユニークなユーザID番号を持つユーザに関する情報を返します。必ずGET USER LIST コマンドによって返された有効なユーザID番号を渡さなければなりません。

ユーザアカウントが存在しない場合や削除された場合、エラーコード -9979が返されます。ON ERR CALL コマンドを用いてインストールされたエラー処理メソッドでこのエラーを検知することができます。その他では、関数 Is user deleted を呼び出してユーザアカウントをテストし、その後このコマンドを呼び出す方法があります。

ユーザID番号は以下の値および範囲を持っています。

ユーザID番号	ユーザ説明
1	設計者ユーザ
2	管理者ユーザ
3 to 15000	データベースの設計者によって作成されたユーザ (ユーザ番号3は、設計者によって最初に作成されたユーザ、ユーザ番号4は、2番目に作成されたユーザなど)
-11 to -15000	データベースの管理者によって作成されたユーザ (ユーザ番号-11は、管理者によって最初に作成されたユーザ、ユーザ番号-12は、2番目に作成されたユーザなど)

呼び出し後、引数 name、startup、password、nbLogin と lastLogin に、ユーザの名前、スタートアップメソッド、暗号化されたパスワード、ログインした回数、最後にログインした日付を返します。

Note: GET USER PROPERTIES コマンドは、引数 password に暗号化されたパスワードをもはや返しません。バージョン6.0.2以降、常に空の文字列がこの引数に返されます。ユーザのパスワードをチェックするには、Validate password 関数を呼び出します。

オプション引数 memberships を渡すと、ユーザが属するグループのユニークなID番号が返されます。グループID番号は以下の値および範囲を持っていいます。

オプション引数 groupOwner を渡すと、ユーザグループ "オーナー" のID番号、つまりこのユーザによって作成されたオブジェクトのデフォルトのオーナーグループを取得します。

グループID番号は以下の値および範囲を持っています。

グループID番号	グループ説明
15001 to 32767	設計者または関連するグループオーナーによって作成されたグループ (グループ番号15001は、設計者によって最初に作成されたグループ、グループ番号15002は、2番目に作成されたグループなど)
-15001 to -32768	管理者または関連するグループオーナーによって作成されたグループ (グループ番号-15001は、管理者によって最初に作成されたグループ、グループ番号-15002は、2番目に作成されたグループなど)

エラー管理

コマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによって既にアクセスされている場合は、アクセス権エラーが生成されます。ON ERR CALL コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

Is license available

Is license available {(license)} -> 戻り値

引数	型	説明
license	倍長整数	→ ライセンスの有効性テストを行うプラグイン
戻り値	ブール	↩ プラグインが利用可能な場合はTrue、その他の場合はFalse

説明

Is license available コマンドを使用して、プラグインの有効性を確認することができます。例えば、プラグインが必要な機能を表示または非表示にする際に有効です。

Is license available コマンドは次のような3通りの使用が可能です。

- 引数 license を省略する場合：4DアプリケーションがデモモードのA場合に、コマンドはFalseを返します。
- 以下の""テーマの定数のうちの1つを引数 license に渡す場合。

定数	型	値
4D Client SOAP License	倍長整数	808465465
4D Client Web License	倍長整数	808465209
4D Draw License	倍長整数	808464694
4D for ADO License	倍長整数	808465714
4D for MySQL License	倍長整数	808465712
4D for OCI License	倍長整数	808465208
4D for PostgreSQL License	倍長整数	808465713
4D for Sybase License	倍長整数	808465715
4D ODBC Pro License	倍長整数	808464946
4D SOAP License	倍長整数	808465464
4D View License	倍長整数	808465207
4D Web License	倍長整数	808464945
4D Write License	倍長整数	808464697

  この方法では、対応するプラグインのライセンスが有効な場合、コマンドは True を返します。コマンドはデザインモード、SET PLUGIN ACCESS コマンドの結果等を考慮に入れて結果を返します。
  プラグインがデモモードで動作している場合、Is license availableはFalse を返します。
- プラグイン "4BNX" リソースのID番号を引数licenseに直接渡す場合、コマンドは、前述の通りに動作します。

Is user deleted

Is user deleted (userNumber) -> 戻り値

引数	型	説明
userNumber	倍長整数	→ ユーザID番号
戻り値	ブール	↩ TRUE = ユーザアカウントが削除されている、または存在しない場合 FALSE = ユーザアカウントがアクティブな場合

説明

Is user deleted コマンドを使用して、引数 userID に渡したユニークなユーザID番号を持つユーザアカウントをテストします。

そのユーザアカウントが存在しない場合や削除されてしまっている場合は、Is user deleted コマンドはTRUEを返します。その他の場合は、FALSEを返します。

エラー管理

コマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによって既にアクセスされている場合は、アクセス権エラーが生成されます。ON ERR CALL コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

Set group properties

Set group properties (groupID ; name ; owner {; members}) -> 戻り値

引数	型	説明
groupID	倍長整数	→ グループのユニークなID番号、または、-1= 設計者グループの追加 -2= 管理者グループの追加
name	文字	→ 新規グループの名前
owner	倍長整数	→ 新規グループオーナーのユーザID番号
members	倍長整数配列	→ 新規グループのメンバー
戻り値	倍長整数	↩ 新規グループのユニークなID番号

説明

Set group properties コマンドを使用すれば、引数 groupID に渡したユニークなグループID番号を持つ既存のグループのプロパティを変更および更新することが可能です。また、設計者あるいは管理者に関連する新規グループを追加することもできます。

既存グループのプロパティを変更している場合は、GET GROUP LIST コマンドによって返された有効なグループID番号を渡さなければなりません。グループID番号は以下の値および範囲を持っています。

グループID番号	グループ説明
15001 to 32767	設計者または関連するグループオーナーによって作成されたグループ (グループ番号15001は、設計者によって最初に作成されたグループ、グループ番号15002は、2番目に作成されたグループなど)
-15001 to -32768	管理者または関連するグループオーナーによって作成されたグループ (グループ番号-15001は、管理者によって最初に作成されたグループ、グループ番号-15002は、2番目に作成されたグループなど)

設計者に関連する新規グループを追加するには、引数 groupID に -1を渡します。管理者に関連する新規グループを追加するには、引数 groupID に -2を渡します。呼び出し後、グループの追加が完了すると、そのユニークなID番号が引数 groupID に返されます。

-1、-2 または有効なグループID番号が渡されない場合、Set group properties コマンドは何も行いません。

呼び出す前に、引数 name と owner にグループの新しい名前とオーナーを渡します。グループのすべてのプロパティを変更したくない場合 (メンバー以外、以下参照) 、最初にGET GROUP PROPERTIES コマンドを呼び出し、変更したくないプロパティに対して返された値を渡します。

オプション引数 members が渡されないと、現在のグループのメンバーリストは変更されません。
グループを追加している際に、引数 members が渡されないと、そのグループにはメンバーが追加されません。

Note: グループオーナーは自身が所有するグループのメンバーとして自動的に設定されるわけではありません。引数 members を使用して、グループオーナーをそのグループに含めるかどうかはユーザ次第です。

オプション引数 members を渡すと、そのグループのメンバーリスト全体を変更します。呼び出す前に、そのグループがメンバーとして取得するユーザとグループのユニークなID番号を配列 members に割り当てなければなりません。

メンバーID番号	**メンバー説明**
1	設計者ユーザ
2	管理者ユーザ
3 to 15000	データベースの設計者によって作成されたユーザ (ユーザ番号3は、設計者によって最初に作成されたユーザ、ユーザ番号4は、2番目に作成されたユーザなど)
-11 to -15000	データベースの管理者によって作成されたユーザ (ユーザ番号-11は、管理者によって最初に作成されたユーザ、ユーザ番号-12は、2番目に作成されたユーザなど)
15001 to 32767	設計者または関連するグループオーナーによって作成されたグループ (グループ番号15001は、設計者によって最初に作成されたグループ、グループ番号15002は、2番目に作成されたグループなど)
-15001 to -32768	管理者または関連するグループオーナーによって作成されたグループ (グループ番号-15001は、管理者によって最初に作成されたグループ、グループ番号-15002は、2番目に作成されたグループなど)

グループからすべてのメンバーを削除するには、空の配列 members を渡します。

エラー管理

コマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによって既にアクセスされている場合は、アクセス権エラーが生成されます。ON ERR CALL コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

SET PLUGIN ACCESS

SET PLUGIN ACCESS (plugIn ; group)

引数	型		説明
plugIn	倍長整数	⇒	プラグイン番号
group	文字	⇒	プラグインに関連するグル—プの名前

説明

SET PLUGIN ACCESS コマンドを使用すれば、データベース上にインストールされた各 "シリアルされた" プラグインをプログラムで設定する環境をユーザグループに提供することができます。

Note: グループエディタを使用して、デザインモードで操作を実行することも可能です。

引数 plugIn にユーザのグループに割り当てられたプラグインの番号を渡します。プラグインのライセンスには4DクライアントWebとSOAPライセンスがあります。“ テーマにある以下の定数の1つを渡します。

定数	型	値
4D Client SOAP License	倍長整数	808465465
4D Client Web License	倍長整数	808465209
4D Draw License	倍長整数	808464694
4D for ADO License	倍長整数	808465714
4D for MySQL License	倍長整数	808465712
4D for OCI License	倍長整数	808465208
4D for PostgreSQL License	倍長整数	808465713
4D for Sybase License	倍長整数	808465715
4D ODBC Pro License	倍長整数	808464946
4D View License	倍長整数	808465207
4D Write License	倍長整数	808464697

プラグインの使用を許可されているユーザが存在しているグループの名前を引数 group に渡します。

Note: 1度にプラグインを使用できるのは、1つのグループだけです。他のグループがプラグインアクセス権を所有している場合、このコマンドを実行すると、そのグループはその権限を失います。

Set user properties

Set user properties (userID ; name ; startup ; password ; nbLogin ; lastLogin {; memberships {; groupOwner}}) -> 戻り値

引数	型		説明
userID	倍長整数	→	ユーザアカウントのユニークなID番号、または -1= Designerに関連したユーザの追加 -2= Administratorに関連したユーザの追加
name	文字	→	新規ユーザの名前
startup	文字	→	新規ユーザスタートアップメソッドの名前
password	文字	→	新しい (暗号化されていない) パスワード、または * を指定すると、パスワードは以前のまま
nbLogin	倍長整数	→	データベースへログインした回数
lastLogin	日付	→	データベースに最後にログインした日付
memberships	倍長整数配列	→	ユーザが属すグループのID番号
groupOwner	倍長整数	→	ユーザーグループオーナーの参照番号
戻り値	倍長整数	↩	新規ユーザのユニークなID番号

説明

Set user properties コマンドを使用すれば、引数 userID に渡したユニークなユーザID番号を持つ既存のユーザアカウントのプロパティを変更および更新することが可能です。また、DesignerあるいはAdministratorに関連する新規ユーザを追加することもできます。

既存のユーザアカウントのプロパティを変更している場合は、GET USER LIST コマンドによって返された有効なユーザID番号を渡さなければなりません。

ユーザアカウントが存在しない場合や削除された場合、エラーコード -9979が返されます。ON ERR CALL コマンドを用いてインストールされたエラー処理メソッドでこのエラーを検知することができます。その他では、関数 Is user deleted を呼び出してユーザアカウントをテストし、その後 Set user properties コマンドを呼び出す方法があります。

ユーザID番号は以下の値および範囲を持っています。

ユーザID番号	ユーザ説明
1	Designerユーザ
2	Administratorユーザ
3 ~ 15000	データベースのDesignerによって作成されたユーザ (ユーザ番号3はDesignerによって最初に作成されたユーザ、ユーザ番号4は、2番目に作成されたユーザなど)
-11 ~ -15000	データベースのAdministratorによって作成されたユーザ (ユーザ番号-11はAdministratorによって最初に作成されたユーザ、ユーザ番号-12は、2番目に作成されたユーザなど)

Designerに関連する新規ユーザを追加するには、引数 userID に -1を渡します。Administratorに関連する新規ユーザを追加するには、引数 userID に -2を渡します。呼び出し後、ユーザの追加および修正が完了すると、そのユニークなID番号が引数 userID に返されます。

-1、-2 または有効なユーザID番号が渡されない場合、Set user properties コマンドは何も行いません。

呼び出す前に、引数 name、startup、password、nbLogin と lastLogin にユーザの新しい名前、スタートアップメソッド、パスワード、ログインした回数と最後にログインした日付を渡します。引数 password に暗号化されていないパスワードを渡すと、4Dはそのパスワードを暗号化し、ユーザアカウントに格納します。
引数 name に渡された新規ユーザの名前がユニークでない場合 (同じ名前を持つユーザが既に存在している) 、コマンドは何も行わず、エラーコード-9979が返されます。ON ERR CALL コマンドを用いてインストールされたエラー処理メソッドでこのエラーを検知することができます。

ユーザのすべてのプロパティを変更したくない場合 (メンバーシップ以外、以下参照) 、最初にGET USER PROPERTIES コマンドを呼び出し、変更したくないプロパティに対して返された値を渡します。

アカウント用のパスワードを変更したくない場合、引数 password の値として、* 記号を渡します。これを実行することにより、アカウント用のパスワードを変更することなく、ユーザアカウントの他のプロパティを変更することができます。

オプション引数 memberships が渡されないと、現在のユーザのメンバーシップは変更されません。
ユーザを追加している際に、引数 memberships が渡されないと、そのユーザはどのグループにも属しません。オプション引数 memberships を渡すと、そのユーザの全てのメンバーシップを変更します。呼び出す前に、そのユーザが属するグループのユニークなID番号を配列 memberships に割り当てなければなりません。

オプション引数 groupOwner を渡すと、ユーザグループ "オーナー" のID番号、つまり、このユーザによって作成されたオブジェクトのデフォルトのオーナーグループを指定します。

グループID番号は以下の値および範囲を持っています。

グループID番号	グループ説明
15001 ~ 32767	Designerまたは関連するグループオーナーによって作成されたグループ (グループ番号15001は、Designerによって最初に作成されたグループ、グループ番号15002は、2番目に作成されたグループなど)
-15001 ~ -32768	Administratorまたは関連するグループオーナーによって作成されたグループ (グループ番号-15001は、Administratorによって最初に作成されたグループ、グループ番号-15002は、2番目に作成されたグループなど)

ユーザのすべてのメンバーシップを無効にするには、空の配列 memberships を渡します。

エラー管理

コマンドを呼び出すための特定のアクセス権を持っていない場合や、パスワードシステムが他のプロセスによって既にアクセスされている場合は、アクセス権エラーが生成されます。ON ERR CALLコマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

User in group

User in group (user ; group) -> 戻り値

引数	型	説明
user	文字	→ ユーザ名
group	文字	→ グループ名
戻り値	ブール	← TRUE = ユーザがグループに存在する場合 FALSE = ユーザがグループに存在しない場合

説明

User in group コマンドはuserがgroupに存在する場合、Trueを返します。

例題

以下の例題を使用して、特定の送り状を探します。カレントユーザがExecutiveグループに属する場合、そのカレントユーザは機密情報を表示するフォームにアクセスすることができます。そのユーザがExecutiveグループに属しない場合、異なるフォームが表示されます。

```
QUERY([Invoices];[Invoices]Retail>100)
If(User in group(Current user;"Executive"))
   FORM SET OUTPUT([Invoices];"Executive Output")
   FORM SET INPUT([Invoices];"Executive Input")
Else
   FORM SET OUTPUT([Invoices];"Standard Output")
   FORM SET INPUT([Invoices];"Standard Input")
End if
MODIFY SELECTION([Invoices];*)
```

USERS TO BLOB

USERS TO BLOB (users)

引数	型		説明
users	BLOB	→	ユーザが存在しなければならないBLOB
		←	ユーザアカウント (暗号化された)

説明

USERS TO BLOB コマンドを使用して、すべてのユーザアカウントのリストと管理者によって作成されたデータベースグループをBLOB users に格納します。

データベース管理者および設計者のみが、このコマンドを実行することができます。他のユーザが実行しようとすると、コマンドは何も行わず、権限エラー (-9949) が生成されます。

生成されたBLOBは自動的に暗号化されます。このBLOBを解読できるのは、BLOB TO USERS コマンドのみです。ハードディスクまたはフィールド上のファイルにこのBLOBを格納します。このコマンドの機能は、ツールボックスからグループとユーザを保存することに相当します。唯一の違いは、このコマンドを使用すると、ファイル内だけでなく、BLOBフィールド上にユーザアカウントを格納することができるということです。

このコンセプトにより、データベースのデータ上でユーザのバックアップを保持することが可能になります。そして、データベースストラクチャファイルを更新する際、バックアップメカニズムおよびユーザを自動的にロードするシステムを実行することができます (ユーザアカウントに関する情報は、4Dによってデータベースストラクチャファイルに格納されます) 。

Validate password

Validate password (userID ; password {; digest}) -> 戻り値

引数	型		説明
userID	倍長整数, 文字	→	ユニークなユーザID
password	文字	→	暗号化されていないパスワード
digest	ブール	→	Digest password = True, Plain-text password (default) = False
戻り値	ブール	↩	True = 有効なパスワード False = 無効なパスワード

説明

Validate password コマンドは引数 password に渡された文字列が、引数 userID に渡されたID番号または名前を持つユーザアカウントのパスワードである場合、Trueを返します。

任意の digest 引数は、password 引数に含まれるパスワードが標準テキストのパスワードかハッシュドパスワード(ダイジェストモード)かを指定します:

- **True** を渡した場合、password にはハッシュドパスワードが格納されていること(ダイジェストモード)を意味します。
- **False** oを渡すかこの引数を省略した場合、password には標準テキストのパスワードが格納されていることを意味します。

この引数はデータベース認証メソッド、特に On REST Authentication データベースメソッド を使用しているときに有用です。

フラッディング (ブルートフォース攻撃) 、言い換えれば複数のユーザ名とパスワードの組み合わせによる試みを防ぐために、コマンドは遅れて実行されます。その結果、このコマンドを4回呼び出すと、10秒間の遅延が発生します。この遅れは、ワークステーション全体を通して発生します。

例題 1

以下の例題を使用して、ユーザ "Hardy" のパスワードが "Laurel" であるかどうかを調べます。

```
GET USER LIST(atUserName;alUserID)
$vlElem:=Find in array(atUserName;"Hardy")
If($vlElem>0)
   If(Validate password(alUserID{$vlElem};"Laurel"))
      ALERT("Yep!")
   Else
      ALERT("Too bad!")
   End if
Else
   ALERT("Unknown user name")
End if
```

例題 2

On REST Authentication データベースメソッド において、接続リクエストを(データベースの4Dユーザーを使用して)テストしたい場合:

```
$0:=Validate password($1;$2;$3)
```

ユーザーインターフェース

- BEEP
- Caps lock down
- Focus object
- GET FIELD TITLES
- GET MOUSE
- GET TABLE TITLES
- HIDE MENU BAR
- INVERT BACKGROUND
- Macintosh command down
- Macintosh control down
- Macintosh option down
- OBJECT Get name
- OBJECT Get pointer
- PLAY
- Pop up menu
- POST CLICK
- POST EVENT
- POST KEY
- REDRAW
- SET ABOUT
- SET CURSOR
- SET FIELD TITLES
- SET TABLE TITLES
- Shift down
- SHOW MENU BAR
- Tool bar height
- Windows Alt down
- Windows Ctrl down
- Get platform interface
- HIDE TOOL BAR
- SET PLATFORM INTERFACE
- SHOW TOOL BAR

BEEP

BEEP
このコマンドは引数を必要としません

説明

BEEP コマンドは、PCまたはMacintoshでビープ音を発生します。発生するビープ音は、サウンドコントロールパネルで変更することができます。

警告: Web接続プロセス内からBEEP コマンドを呼び出さないでください。これは、クライアントであるWebブラウザマシンではなく、Webサーバマシン上の4Dでビープ音が発生するためです。

例題

以下の例は、クエリでレコードが見つからなかった場合に、ビープ音を発生し、警告が表示されます。

```
QUERY([Customers];[Customers]Name=$vsNameToLookFor)
If(Records in selection([Customers])=0)
   BEEP
   ALERT("There is no Customer with such a name.")
End if
```

Caps lock down

Caps lock down -> 戻り値

引数	型	説明
戻り値	ブール	↩ Caps Lockキーの状態

説明

Caps lock down はCaps Lock キーが押されているとTrueを返します。

例題

Shift down コマンドの例を参照してください。

Focus object

Focus object -> 戻り値

引数	型		説明
戻り値	ポインター	←	フォーカスを持つオブジェクトへのポインタ

互換性に関する注意

このコマンドは互換性の目的で保持されています。4D v12からはOBJECT Get pointerコマンドの利用が推奨されます。

説明

Focus object はカレントフォーム中でフォーカスを持つオブジェクトへのポインタを返します。フォーカスを持つオブジェクトがない場合、コマンドはNilを返します。を使用して、どのオブジェクトが現在選択されているかを知る必要なく、フォームエリア上でアクションを実行できます。アクションを実行する前にTypeコマンドを使用して、オブジェクトが正しいデータ型かを確認してください。

Note: Focus objectがリストボックスで使用された場合、コマンドはコンテキストに応じてリストボックスまたはリストボックスの列へのポインタを返します。詳細はPRINT LABELContextual clickを参照してください。

このコマンドはサブフォーム中のフィールドで使用することはできません。

Note: このコマンドはデータ入力のコンテキストのみで使用できます。そうでなければエラーが返されます。

例題

以下の例題はボタンのオブジェクトメソッドです。このオブジェクトメソッドはカレントのオブジェクトのデータを大文字に変更します。オブジェクトのデータ型はテキストまたは文字列でなければなりません (タイプ 0 または 24):

```
$vp :=Focus object ` 現在のエリアのポインタを取得
Case of
   :(Nil($pointer)) ` オブジェクトにフォーカスがない
      ...
   :((Type($vp->)=Is Alpha Field)|(Type($vp->)=Is Text)) ` 文字列フィールドまたはテキストなら
      $vp->:=Uppercase($vp->) ` 大文字にする
End case
```

GET FIELD TITLES

GET FIELD TITLES (aTable ; fieldTitles ; fieldNums)

引数	型		説明
aTable	テーブル	→	フィールド名を取得したいテーブル
fieldTitles	テキスト配列	←	カレントフィールドの名前
fieldNums	倍長整数配列	←	フィールド番号

説明

GET FIELD TITLES コマンドを使用して、目的の table に対してデータベースフィールドの名前と番号を配列 fieldTitles と fieldNums に受け取ります。これら2つの配列内容は同期化されています。

セッション中に SET FIELD TITLES コマンドが呼び出されると、GET FIELD TITLES コマンドは "修正された" 名前と、このコマンドによって定義されたフィールド番号のみを返します。そうでない場合、GET FIELD TITLES コマンドはストラクチャウィンドウで定義されているすべてのデータベースフィールドの名前を返します。
これら両方の場合で、コマンドは非表示フィールドを返しません。

GET MOUSE

GET MOUSE (mouseX ; mouseY ; mouseButton {; *})

引数	型		説明
mouseX	倍長整数	←	マウスの水平座標
mouseY	倍長整数	←	マウスの垂直座標
mouseButton	倍長整数	←	マウスボタンのステータス 0 = 何もしていない 1 = ボタンの押下 2 = 右マウスボタンの押下 3 = 両方のボタンの押下
*	演算子	→	指定した場合、グローバルの座標システムが使用される。省略した場合、ローカルの座標システムが使用される

説明

GET MOUSE コマンドは、マウスの現在の状態を返します。

水平座標と垂直座標が引数 mouseX と mouseY に返されます。オプション引数 * を渡すと、これらの座標は画面に対して相対的に表されます。引数 * を省略すると、座標はカレントプロセスの最前面のウィンドウに対して相対的に表されます。

引数 mouseButton は、上記のようにボタンの状態を返します。

Note: バージョン10.2.5 以降の Mac OS X のみ、値2と3が返されます。

例題

Pop up menu コマンドの例題を参照してください。

GET TABLE TITLES

GET TABLE TITLES (tableTitles ; tableNums)

引数	型	説明
tableTitles	テキスト配列	← カレントテーブルの名前
tableNums	倍長整数配列	← テーブル番号

説明

GET TABLE TITLES コマンドを使用して、ストラクチャウィンドウで、またはSET TABLE TITLES コマンドを用いて定義されたデータベーステーブルの番号と名前を配列 tableTitles と tableNums に取得します。これら2つの配列の内容は同期化されています。

セッション中に SET TABLE TITLES コマンドが呼び出されると、GET TABLE TITLES コマンドは "修正された" 名前とこのコマンドによって定義されたテーブル番号のみを返します。その他の場合、GET TABLE TITLES コマンドは、ストラクチャウィンドウで定義されているすべてのデータベーステーブルの名前を返します。
これら両方の場合で、コマンドは非表示テーブルを返しません。

HIDE MENU BAR

HIDE MENU BAR
このコマンドは引数を必要としません

説明

HIDE MENU BARコマンドは、メニューバーを隠します。

既にメニューバーが隠れている場合は、このコマンドは何も行いません。

例題

以下のメソッドは、マウスボタンをクリックするまでフルスクリーン表示（Macintosh）でレコードを表示します。

```
HIDE TOOL BAR
HIDE MENU BAR
Open window(-1;-1;1+Screen width;1+Screen height;Alternate dialog box)
FORM SET INPUT([Paintings];"Full Screen 800")
DISPLAY RECORD([Paintings])
Repeat
   GET MOUSE($vlX;$vlY;$vlButton)
Until($vlButton#0)
CLOSE WINDOW
SHOW MENU BAR
SHOW TOOL BAR
```

Note: Windows上では、ウィンドウはアプリケーションウィンドウの範囲内に制限されます。

INVERT BACKGROUND

INVERT BACKGROUND ({* ;} textVarﾊ|ﾊtextField)

引数	型	説明
*	演算子	➡変数またはオブジェクト名を指定
textVarﾊ|ﾊtextField	変数, フィールド	➡反転表示される変数またはフィールド

説明

INVERT BACKGROUND コマンドは、フォーム中の textVar または textField を反転表示するために使用します。

このコマンドの有効範囲は使用中のフォームです。

このコマンドは、画面への表示やドットマトリックスプリンタへの印刷を実行する際に使用します。ポストスクリプトプリンタは反転させた表示を印刷することはできません。

出力フォーム内の変数を反転することはできません。INVERT BACKGROUND コマンドは、入力可能な変数に対して使用しないようにしてください。文字を入力すると、反転させた表示は部分的に消去されます。

例題

以下の例は、フィールドの値を判定する入力フォーム内の変数のオブジェクトメソッドです。このオブジェクトメソッドは、フィールドの値が正の場合は何も実行せず、負の場合はフォーム内にある変数の表示を反転します。

```
vAmount:=[Accounts]Amount `変数にフィールドの値を代入する
If(vAmount<0) `変数の値が負の場合
   INVERT BACKGROUND(vAmount) `変数を反転表示にする
End if
```

Note: このコマンドは本来、白黒のユーザインタフェース用に作成されましたが、現在では、ほとんど使用されていません。一般的に、フィールドや変数を反転するにはカラーを使用します。

Macintosh command down

Macintosh command down -> 戻り値

引数	型		説明
戻り値	ブール	↩	MacintoshのCommandキーのステータス (Windowsの場合は、Ctrlキー)

説明

Macintosh command downはMacintoshのcommandキーが押されているとTrueを返します。

Note: Windowsのプラットフォーム上で呼び出された場合は、WindowsのCtrlキーが押されていると、Macintosh command down はTRUEを返します。

例題

Shift down コマンドの例を参照してください。

Macintosh control down

Macintosh control down -> 戻り値

引数	型		説明
戻り値	ブール	↩	MacintoshのControlキーのステータス

説明

Macintosh control down コマンドはMacintoshのControlキーが押されているとTrueを返します。

Note: Windowsのプラットフォーム上で呼び出された場合は、Macintosh control down コマンドは常にFalseを返します。このMacintosh用の同等のキーは、Windows上にありません。

例題

Shift down コマンドの例題を参照してください。

Macintosh option down

Macintosh option down -> 戻り値

引数	型		説明
戻り値	ブール	←	Macintosh Optionキーの状態 (Windows上ではAltキー)

説明

Macintosh option down はMacintoshのoptionキーが押されているとTrueを返します。

Note: WindowsのプラットフォームЕ上で呼び出された場合は、WindowsのAltキーが押されていると、Macintosh option down はTrueを返します。

例題

Shift down コマンドの例を参照してください。

OBJECT Get name

OBJECT Get name {(selector)} -> 戻り値

引数	型		説明
selector	倍長整数	→	オブジェクトカテゴリ
戻り値	テキスト	↩	オブジェクトの名前

説明

OBJECT Get name コマンドはフォームオブジェクトの名前を返します。

コマンドを使用して、selector引数の値に基づき、2タイプのオブジェクトを指定できます。この引数には以下の定数を渡せます (テーマ):

- Object current (selector 省略時のデフォルト): このセレクタを渡すかselector引数を省略した場合、コマンドはコマンドを呼び出した (オブジェクトメソッドあるいはオブジェクトメソッドから呼ばれたサブメソッド) オブジェクトの名前を返します。この場合、コマンドはフォームオブジェクトのコンテキストから呼ばれなければなりません。そうでなければ空の文字列を返します。
- Object with focus: このセレクタを渡した場合、コマンドはフォーム上でフォーカスを持つオブジェクトの名前を返します。

例題

"bValidateForm" ボタンのオブジェクトメソッド:

```
$btnName:=OBJECT Get name(Object current)
```

このオブジェクトメソッド実行後、*$btnName*変数には"bValidateForm"が格納されています。

OBJECT Get pointer

OBJECT Get pointer ({selector }{;}{ objectName {; subformName}}) -> 戻り値

引数	型		説明
selector	倍長整数	→	オブジェクトカテゴリ
objectName	テキスト	→	オブジェクト名
subformName	テキスト	→	サブフォームオブジェクト名
戻り値	ポインター	↩	オブジェクト変数へのポインタ

説明

OBJECT Get pointerコマンドはフォームオブジェクトの変数へのポインタを返します。

このコマンドを使用して、selector引数の値に基づき、異なるオブジェクトを指定できます。この引数にはForm Objects (Access)テーマの定数を渡します:

- Object current (selector省略時のデフォルト): このセレクタを渡すかselector引数を省略した場合、コマンドはカレントオブジェクト (メソッドを実行しているオブジェクト) に割り当てられた変数へのポインタを返します。
  注: これは以前のSelfコマンドとまったく同じ動作です。Selfコマンドは互換性の目的でのみ保持されています。
- Object with focus: このセレクタを渡すと、コマンドはフォーム内でフォーカスを持つオブジェクトに割り当てられた変数へのポインタを返します。残り2つのオプション引数は渡されても無視されます。
  注: これは完全にFocus objectコマンドと同じ動作です。Focus objectコマンドは4D v12で廃止予定となりました。
- Object subform container: このセレクタを渡すと、コマンドはサブフォームコンテナにバインドされた変数へのポインタを返します。残り2つのオプション引数は渡されても無視されます。つまりこのセレクタは、コンテナオブジェクトにバインドされた変数にアクセスするために、サブフォームとして使用されるフォームのコンテキストでのみ利用できます。
- Object named: このセレクタを渡す場合、2番目のobjectName引数も渡さなければなりません。この場合、コマンドはこの引数に渡された名前を持つオブジェクトに割り当てられた変数へのポインタを返します。
  注:objectName がサブフォームに対応し、"一覧サブフォーム"オプションがチェックされていると、コマンドはソーステーブルが指定されていればサブフォームのテーブルへのポインタを返します。そうでなければNilを返します。

オプションのsubformName引数を使用してカレントのコンテキスト (すなわち親フォーム) に属さないobjectNameオブジェクトへのポインタを取得できます。この引数を使用可能にするには、Object named セレクタを使用しなければなりません。
subformName引数が渡されると、**OBJECT Get pointer**コマンドはまずカレントフォーム内でsubformNameという名称のサブフォームを探し、そしてその中でobjectNameという名称のオブジェクトを探します。このオブジェクトが見つかると、このオブジェクトの変数へのポインタを返します。

例題

同じ親フォーム上でサブフォームとして2回使用される"SF"フォームがあります。サブフォームオブジェクトにはそれぞれ"SF1"と"SF2"という名前が与えられます。"SF"フォームには*CurrentValue*という名称のオブジェクトがあります。親フォームのフォームメソッドの"On Load"フォームイベントで、SF1の*CurrentValue*オブジェクトを"January"に、SF2のそれを"February"に初期化します:

```
C_POINTER($Ptr)
$Ptr:=OBJECT Get pointer(Object named;"CurrentValue";"SF1")
$Ptr->:="January"
$Ptr:=OBJECT Get pointer(Object named;"CurrentValue";"SF2")
$Ptr->:="February"
```

PLAY

PLAY (objectName {; channel})

引数	型		説明
objectName	文字	→	サウンドファイル名、またはMac OSの"snd "リソース、またはシステムサウンド 非同期再生を中断する場合、空の文字列
channel	倍長整数	→	指定した場合、シンセサイザーチャネルと非同期実行、省略した場合同期実行

説明

PLAYコマンドを使用してサウンドやマルチメディアファイルを再生できます。再生したいファイルの完全なパス名を objectName に渡します。Mac OSでは、コマンドを使用してサウンドリソースやシステムサウンドを再生することもできます。

- ファイルを再生するには、objectNameにそのパス名を渡します。フルパス名またはデータベースストラクチャファイルからの相対パス名を渡せます。主なサウンドおよびマルチメディアフォーマットがサポートされています: .WAV, .MP3, .AVI, .AIFF (Mac OS), 等。Mac OSでは、コマンドは特にCore Audioフォーマットをサポートしています。
  注: 非同期モードでマルチメディアファイルやオブジェクトを再生することはできません。そうしたい場合はOLEサービスを使用します。
- (Mac OSのみ) システムサウンドを再生するには、objectName 引数に直接その名前を渡します。
  注: 互換性のため、4Dはまず'snd 'リソースからobjectNameを探します。

引数 channel は、Macintoshシンセサイザチャネルを指定します。channel が指定されていない場合、 channel はデジタル化された単純なサウンドとみなされ、同期指定になります。同期指定では、サウンドが終了するまですべての処理が停止します。channel が0の場合、チャネルはデジタル化された単純なサウンドとみなされ、非同期指定になります。非同期指定では、処理は停止せず、サウンドがバックグラウンドで再生します。

非同期サウンドの再生を停止するには、以下のステートメントを使用します。

```
PLAY("";0)
```

例題 1

Windows上でのビデオファイルの再生方法を以下の例題で示します。

```
$DocRef :=Open document("";"AVI")
If(OK=1)
   CLOSE DOCUMENT($DocRef)
   PLAY(Document)
End if
```

例題 2

以下のコードがstartupメソッドにあります。Macintosh上では、ウエルカムの音でユーザを歓迎します。

```
PLAY("Welcome Sound") `ウエルカムの音を再生する
```

Pop up menu

Pop up menu (contents {; default {; xCoord ; yCoord}}) -> 戻り値

引数	型		説明
contents	テキスト	→	定義された項目テキスト
default	倍長整数	→	デフォルトで選択された項目番号
xCoord	倍長整数	→	左上角のX座標
yCoord	倍長整数	→	左上角のY座標
戻り値	倍長整数	↩	選択された項目番号

説明

Pop up menu コマンドは、現在マウスが置かれている場所でポップアップメニューを表示します。

ユーザインタフェースの規則に従い、通常マウスボタンが押されて、かつ押されたままのときにこのコマンドを呼び出します。

以下のように、引数contentsを用いてポップアップメニューの項目を定義します。

- 各項目をセミコロン (;) で区切ります。例えば、"**ItemText1;ItemText2;ItemText3**" のようになります。
- 項目を無効にするには、項目テキスト内に開いたカッコ (() を指定します。
- 区切り線を指定するには、項目テキストとして "-" または "(-" を渡します。
- 行のフォントスタイルを指定するには、項目テキスト内に (<) を指定し、続けて以下の文字の1つを記述します。

  <B 太字

  <I 斜体

  <U アンダーライン

  <O アウトライン (Macintoshのみ)

  <S シャドウ (Macintoshのみ)

- 項目にチェックマークをを付けるには、項目テキスト内に疑問符 (!) を指定し、続けてチェックマークとして表示したい文字を記述します。
  - Macintoshでは、文字は直接表示されます。システムバージョンおよびランゲージにおいて、標準チェックマークを表示するには、次のステートメントを使用します: Char(18)
  - Windowsでは、いかなる文字を渡してもチェックマークが表示されます。
- 項目にアイコンを付けるには、項目テキスト内にアクセント (^) を置き、続けてMac OSベースのアイコンリソースのリソースIDであるコード +208 を持つ文字を指定します。
- 項目にショートカットを付けるには、項目テキスト内にスラッシュを (/) 設定し、続けて項目のためのショートカット文字を指定します。この最後のオプションは、純粋に情報提供のためのものであることに注意してください。ショートカットがポップアップメニューをアクティブにすることはありません。ただし、アプリケーションのメインメニューバーにそのポップアップメニュー項目に相当するのものがある場合にショートカットを情報として含めることができるということです。

Tip: 特殊文字 (!, /, etc.) をポップアップメニューに含めるなどのために、ポップアップメニューでこれらの文字を解釈するためのメカニズムを無効にすることが可能です。これを実行するには引数contentsをChar(1)で始め、さらにこのステートメントを区切り文字として使用します。

```
contents:=Char(1)+"1/4"+Char(1)+"1/2"+Char(1)+"3/4")
```

このステートメントが実行されると、スタイルおよびショートカットをポップアップメニューに割り当てることができなくなりますので注意してください。

オプション引数 default によって、ポップアップメニューが表示される際にデフォルトで選択されるメニュー項目を指定できます。1からメニュー項目の数までの値を渡します。この引数を省略すると、コマンドはデフォルトで最初のメニュー項目を選択します。

オプション引数 xCoord と yCoord を用いて、表示されるポップアップメニューの位置を指定できます。 xCoord と yCoord には、メニューの左上角の水平座標と垂直座標をそれぞれ渡します。これらの座標は、カレントフォームのローカル座標システムにおいてピクセル形式で表示されなければなりません。これら2つの引数は必ず一緒に渡さなければなりません。1つのみが渡された場合、それは無効となります。
引数 xCoord と yCoord を使用すると、引数 default は無視されます。この場合、マウスは必ずしもポップアップメニューのレベルに置かれてはいません。
これらの引数は、特に、ポップアップメニューと連動する3Dボタンを扱う際に便利です。

メニュー項目を選択すると、コマンドはその番号を返します。その他の場合はゼロ (0) を返します。

Note: 適切な項目数のポップアップメニューを使用してください。50以上もの項目を表示したい場合は、ポップアップメニューではなく、フォーム内のスクロール可能エリアの使用を検討するほうが賢明です。

例題

プロジェクトメソッドMY SPEED MENUは、ナビゲーションスピードメニューをプルダウンします。

```
  ` MY SPEED MENU プロジェクトメソッド
 GET MOUSE($vlMouseX;$vlMouseY;$vlButton)
 If(Macintosh control down|($vlButton=2))
    $vtItems:="About this database...<I;(-;!-Other Options;(-"
    For($vlTable;1;Get last table number)
       If(Is table number valid($vlTable))
          $vtItems:=$vtItems+";"+Table name($vlTable)
       End if
    End for
    $vlUserChoice:=Pop up menu($vtItems)
    Case of
       :($vlUserChoice=1)
  ` Display Information
       :($vlUserChoice=2)
  `オプションを表示する
       Else
          If($vlUserChoice>0)
  `番号が $vlUserChoice-4 のテーブルに移動する
          End if
    End case
 End if
```

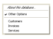
このプロジェクトメソッドは、以下から呼び出せます。

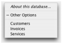
- マウスボタンがリリースされるのを待たずにマウスクリックに反応するフォームオブジェクトのメソッド (透明ボタン)
- イベントを "スパイ" し、他のプロセスと通信するプロセス
- ON EVENT CALL を用いてインストールされたイベント処理メソッド

最後の2つの場合では、フォームオブジェクトでクリックが行われる必要はありません。これは、Pop up menu コマンドの1つの利点です。一般的にポップアップメニューを表示するためにフォームオブジェクトを使用します。Pop up menu を使用すると、どこにでもメニューを表示することができます。

Windowsでは、マウスの右ボタンを押すことによってポップアップメニューが表示されます。Macintoshでは、Control+ クリックを押すことによって表示されます。ただし、メソッドはマウスクリックが存在するかどうかを実際にチェックをしませんので注意してください。呼び出し側のメソッドがこのテストを実行します。

Windows上 (左)とMacintosh上 (右)で表示されるポップアップメニューは以下のとおりです。Windowsバージョンの標準チェックマークに注目してください。





POST CLICK

POST CLICK (mouseX ; mouseY {; process} {; *})
mouseX 倍長整数→水平座標
mouseY 倍長整数→垂直座標
process 倍長整数→送信先のプロセス参照番号、省略または0の場合、アプリケーションイベントキュー
* →指定された場合はグローバルな座標システムが使用される 省略された場合はローカルな座標システムが使用される

説明

POST CLICK コマンドはマウスクリックをシミュレートします。これは、ユーザが実際にマウスボタンをクリックした場合と同様の結果を生じます。

引数 mouseX と mouseY には、クリックの水平座標と垂直座標を渡します。引数 * を渡した場合、これらの座標は画面にに対して相対的に表されます。引数 * を省略すると、process に渡したプロセス番号を持つプロセスの最前面のウィンドウに対して相対的に表されます。

引数 process を指定すると、クリックは process に渡したプロセス番号を持つプロセスへ送られます。0 (ゼロ) を渡したりこの引数を省略すると、クリックはアプリケーションレベルに送られます。そして、4Dスケジューラーがそれを適切なプロセスにディスパッチします。

POST EVENT

POST EVENT (what ; message ; when ; mouseX ; mouseY ; modifiers {; process})

引数	型	説明
what	倍長整数	⇒イベントのタイプ
message	倍長整数	⇒イベントメッセージ
when	倍長整数	⇒Tick単位でのイベント時間
mouseX	倍長整数	⇒マウスの水平座標
mouseY	倍長整数	⇒マウスの垂直座標
modifiers	倍長整数	⇒モディファイアキーのステータス
process	倍長整数	⇒送信先のプロセス参照番号、省略または0の場合、アプリケーションイベントキュー

説明

POST EVENT コマンドは、キーボードまたはマウスイベントをシミュレートします。これは、ユーザが実際にキーボードやマウス上で動作を行った場合と同様の結果を生じます。

引数 what には、以下の値のいずれかを渡します。

定数	型	値
Auto key event	倍長整数	5
Key down event	倍長整数	3
Key up event	倍長整数	4
Mouse down event	倍長整数	1
Mouse up event	倍長整数	2

イベントがマウス関連のイベントであれば、0 (ゼロ) を message に渡します。イベントがキーボード関連のイベントであれば、シミュレートされる文字のコードを message に渡します。

通常、Tickcount によって返される値を when に渡します。

イベントがマウス関連のイベントであれば、クリックの水平座標と垂直座標を mouseX と mouseY に渡します。

引数 modifiers には、テーマの定数を1つまたは組み合わせを渡します。

定数	型	値	コメント
Activate window bit	倍長整数	0	
Activate window mask	倍長整数	1	
Caps Lock key bit	倍長整数	10	
Caps Lock key mask	倍長整数	1024	
Command key bit	倍長整数	8	
Command key mask	倍長整数	256	Windows = Ctrlキー、Mac OS = Commandキー
Control key bit	倍長整数	12	
Control key mask	倍長整数	4096	Mac OSのみ
Mouse button bit	倍長整数	7	
Mouse button mask	倍長整数	128	
Option key bit	倍長整数	11	
Option key mask	倍長整数	2048	Windows = Altキー、Mac OS = Optionキー
Right control key bit	倍長整数	15	
Right control key mask	倍長整数	32768	
Right option key bit	倍長整数	14	
Right option key mask	倍長整数	16384	
Right shift key bit	倍長整数	13	
Right shift key mask	倍長整数	8192	
Shift key bit	倍長整数	9	
Shift key mask	倍長整数	512	WindowsおよびMac OS

例えば、Shift キーをシミュレートするには Shift key bit を渡します。

引数 process を指定すると、イベントは、process に渡したプロセス番号を持つプロセスへと送られます。0 (ゼロ) を渡したり、この引数を省略すると、イベントはアプリケーションレベルに送られます。そして、4Dスケジューラーがそれを適切なプロセスにディスパッチします。

⚙POST KEY

POST KEY (code {; modifiers {; process}})

引数	型		説明
code	倍長整数	⇒	文字コードまたはファンクションキーコード
modifiers	倍長整数	⇒	モディファイアキーのステータス
process	倍長整数	⇒	送信先のプロセス参照番号、省略または0の場合、アプリケーションイベントキュー

説明

POST KEY コマンドはキーストロークをシミュレートします。これは、ユーザが実際にキーボード上で文字を入力した場合と同様の結果を生じます。

引数 code には、その文字のコードを渡します。

引数 modifiers を渡す場合は、イベント (モディファイア) 定数の1つまたは組み合わせを渡します。例えば、Shiftキーをシミュレートするには、Shift key mask を渡します。引数 modifiers を渡さなかった場合は、モディファイアはシミュレートされません。

引数 process を指定すると、キーストロークは、process に渡したプロセス番号を持つプロセスへと送られます。0 (ゼロ) を渡したり、この引数を省略すると、キーストロークはアプリケーションレベルに送られます。そして、4Dスケジューラーはそれを適切なプロセスにディスパッチします。

例題

Process number コマンドの例題を参照してください。

REDRAW

REDRAW (object)

引数	型		説明
object	フォームオブジェクト	→	サブフォームを再描画するテーブル、またはエリアを再描画するフィールド、またはエリアを再描画する変数、または更新されるべきリストボックス

説明

メソッドを使用して、サブフォームで中に表示されるフィールドの値を変更する場合、フォームを確実に更新するために REDRAW コマンドを使用しなければなりません。

セレクション表示モードのリストボックスのコンテキストでは、REDRAWがリストボックスタイプのオブジェクトに適用されると、オブジェクトに表示されているデータが再描画されます。このステートメントは特にセレクションのレコードに対し、データの更新が行われた場合に呼び出されなければなりません。

SET ABOUT

SET ABOUT (itemText ; method)

引数	型		説明
itemText	文字	⇒	アバウトメニュー項目の新しいテキスト
method	文字	⇒	メニューが選択された時に実行するメソッドの名前

説明

SET ABOUTコマンドは、ヘルプ (Windows) またはアプリケーション (Mac OS X) 内の"4Dについて"メニューをitemTextに変更します。

実行後、ユーザがこのメニューをデザインまたはアプリケーションモードで選択すると、methodが実行されます。一般的に、このメソッドではデータベースに関するバージョン情報を示すダイアログボックスを表示します。

このコマンドは4D (すべてのモード)、4D Desktop、および4D Serverで使用されます。サーバマシンで実行されると、新しいプロセス上で実行されます。

例題 1

以下の例は4DについてメニューをAbout Schedulerに置き換えます。ABOUTメソッドはカスタマイズされたアバウトボックスを表示します:

```
SET ABOUT(About Scheduler;ABOUT)
```

例題 2

以下の例題は4Dについてメニューをリセットします:

```
SET ABOUT("4Dについて";"")
```

SET CURSOR

SET CURSOR {(cursor)}

引数	型		説明
cursor	倍長整数	→	Mac OS ベースのカーソルリソース番号

互換性に関するメモ

このコマンドは廃止されたMac OSの'CURS'リソースメカニズムに基づいており、互換性の目的でのみ保持されています。新規の開発ではこのコマンドの利用は推奨されません。

説明

SET CURSORコマンド はマウスカーソルを引数 cursor に渡したID番号を持つMac OS ベースの'CURS' リソースに格納されたカーソルに変更します。

引数を省略すると、マウスカーソルは標準の矢印に設定されます。

以下は、引数 cursor に渡すことのできる基本カーソルです。

1.
2.
3.
4.

SET FIELD TITLES

SET FIELD TITLES (aTable ; fieldTitles ; fieldNumbers {; *})

aTable	テーブル	⇒フィールドタイトルを設定するテーブル
fieldTitles	文字配列	⇒ダイアログボックスに表示するフィールドの名前
fieldNumbers	倍長整数配列	⇒実際のフィールド番号
*		⇒カスタマイズした名前をフォーミュラエディタで使用する

説明

SET FIELD TITLES コマンドを使用すれば、アプリケーションモードにおいて、クエリのような標準の4 Dダイアログボックスを表示する際に、aTable や aSubtable に渡されるそのテーブルのフィールドのマスクや名前の変更、並べ替えが行えます (特に4Dのランゲージコマンド経由でエディタが呼び出された際) 。
このコマンドを使用すると、動作中にフォーム上のフィールド名のラベルを変更することが可能です。これには、ダイナミックな名前を使用します。ダイナミックなテーブル名とフィールド名の挿入についての詳細は、4D Design Reference マニュアルを参照してください。

引数 fieldTitles と fieldNumbers 配列は同期させる必要があります。
配列 fieldTitles には、表示させたいフィールドの名前を渡します。
ある特定のフィールドを表示したくない場合には、そのフィールド名または新しく付けたタイトルを配列に含めないようにします。フィールドは、この配列に指定した順序で表示されます。
配列 fieldNumbers の各要素には配列 fieldTitles の同じ要素の数値に渡したフィールド名または新しいタイトルに対応する実際のフィールド番号を渡します。

例えば、フィールドF、G、Hで構成されるテーブルまたはサブテーブルがあり、フィールドはこの順序で作成されたとします。表示の際には、これらのフィールドをM、N、Oという名前にし、さらにフィールドNは表示したくないとします。最終的に、OとMを、この順序で表示することにします。この場合、配列 fieldTitles の2つの要素としてOとMを渡し、配列 fieldNumbers の2つの要素として3と1を渡します。

オプションの * 引数は、カスタマイズした名前を4Dフォーミュラで使用するかを指定します。

- 省略した場合、カスタマイズした名前は使用されず、デフォルトでフィールド定義時の実際のフィールド名が使用されます。ランゲージインタープリターはカスタム名を処理しないため、カスタム名を利用することは、フィールドの命名に多大な自由度を与えます。
- * 引数を渡した場合、このコマンドで指定したカスタマイズした名前を4Dフォーミュラで使用されます。ただし、4D ランゲージインタプリタが禁止している文字(例えば、-?*!)をカスタム名に含めることはできません (詳細は識別子を参照) 。

Note: フォーミュラエディタレベルでは、* 引数を省略してコマンドを呼び出しても、以前に * 引数付きでコマンドを呼び出したときの設定を変更しません。言い換えれば、フォーミュラエディタは常に、最後に * 引数付きで呼び出された設定が有効となります。

SET FIELD TITLES コマンドは、テーブルの実際のストラクチャを変更するわけではありません。ランゲージコマンドで標準の4Dダイアログボックスやダイナミックな名前を使用しているフォームを呼び出したときのみ効果があります (デザインモードにおいて、エディタまたはフォームがメニューコマンドから呼び出された場合、データベースの実際のストラクチャが表示されます) 。SET FIELD TITLES コマンドの有効範囲は、そのワークセッションです。クライアント/サーバにおける利点の1つは、複数の4Dリモートステーションがそれぞれ異なる見方でサーバのテーブルを同時に "見る" ことができる点です。SET FIELD TITLES コマンドは、何度でも呼び出せます。

SET FIELD TITLES コマンドは、以下のような場合に使用します。

- テーブルを動的にローカライズする。
- フィールド表示を、実際のテーブル定義に関係なく、独自のものとする。
- フィールド表示を、ユーザー固有のものや権限によるものとする。

警告:

- SET FIELD TITLES コマンドはフィールドの非表示属性を上書きしません。データベースの設計レベルでフィールドが非表示として設定されている場合、SET FIELD TITLES コマンドへの呼び出しにそのフィールドを指定しても、アプリケーションモードにおいてフィールドは表示されません。
- プラグインは常にコマンドによって指定された仮想ストラクチャへアクセスします。

例題

SET TABLE TITLES コマンドの例題を参照してください。

SET TABLE TITLES

SET TABLE TITLES (tableTitles ; tableNumbers {; *})

tableTitles	文字配列	⇒ダイアログボックスに表示されるテーブル名
tableNumbers	倍長整数配列	⇒実際のテーブル番号
*		⇒フォーミュラエディタでカスタム名を使用

説明

SET TABLE TITLESコマンドを使用すれば、アプリケーションモードにおいて、標準の4Dダイアログボックスに現れるデータベーステーブルを一部隠したり、表示名称を変更したり、並び順を変えたりできます (ただしこれらのダイアログが4Dランゲージコマンドから呼び出された場合)。例えばこのコマンドを使用すれば、アプリケーションモードでのクエリエディターのテーブル表示を変更できます。

さらにこのコマンドを使用すると、動作中にフォーム上のテーブル名のラベルを変更することが可能です。これには、ダイナミックな名前を使用します。ダイナミックなテーブル名とフィールド名の挿入についての詳細は、4Dデザインリファレンスマニュアルのスタティックテキスト中で参照を使用するを参照してください。

引数 tableTitles と tableNumbers 配列は同期させる必要があります。配列 tableTitles には、表示させたいテーブルの名前を渡します。

ある特定のテーブルを表示したくない場合には、そのテーブル名または新しく付けたタイトルを、配列含めないようにします。テーブルはこの配列に指定した順序で表示されます。

配列 tableNumbers の各要素には、配列 tableTitles にある同じ要素の数値に渡したテーブル名または新しいタイトルに対応する実際のテーブル番号を渡します。

例えば、テーブルA、B、Cで構成されるデータベースがあり、テーブルはこの順序で作成されたとします。表示の際には、これらのテーブルをX、Y、Zという名前にし、さらにテーブルBは表示したくないとします。最終的に、ZとXを、この順序で表示することにします。この場合、配列 tableTitles の2つの要素としてZとXを渡し、配列 tableNumbers の2つの要素として3と1を渡します。

オプションの * 引数は、カスタマイズした名前を4Dフォーミュラで使用するかを指定します。

- 省略した場合カスタマイズした名前は使用されず、デフォルトで実際のテーブル名が使用されます。ランゲージインタープリターはカスタム名を処理しないため、カスタム名を利用することは、テーブルの命名に多大な自由度を与えます。
- * 引数を渡した場合は、このコマンドで指定した名前を4Dフォーミュラで使用されます。**ただし**、4D ランゲージインタプリタが禁止している文字 (例えば、-?*!)をカスタム名に含めることはできません。例えばフォーミュラで"Rate_in_%"のような名前を使用することはできません (詳細は識別子を参照)。

Note: フォーミュラエディタレベルでは、* 引数を省略してコマンドを呼び出しても、以前に * 引数付きでコマンドを呼び出したときの設定を変更しません。言い換えれば、フォーミュラエディタは常に、最後に * 引数付きで呼び出された設定が有効となります。

SET TABLE TITLES コマンドは、データベースの実際のストラクチャを変更するわけではありません。ランゲージコマンド経由で標準の4Dダイアログボックスやダイナミックな名前を使用しているフォームを呼び出したときのみ影響します (デザインモードにおいて、エディタまたはフォームがメニューコマンドから呼び出された際、データベースの実際のストラクチャが表示されます) 。**SET TABLE TITLES**コマンドの有効範囲は、そのワークセッションです。クライアント/サーバにおける利点の1つは、複数の4Dクライアントステーションがそれぞれ異なる見方でデータベースを同時に "見る" ことができる点です。**SET TABLE TITLES** コマンドは、何度でも呼び出せます。

SET TABLE TITLESコマンドは、以下のような場合に使用します。

- データベースを動的にローカライズする。
- テーブル表示を、実際のデータベース定義に関係なく、独自のものとする。
- テーブル表示を、ユーザー固有のものや権限によるものとする。

注:

- **SET TABLE TITLES**コマンドはフィールドの非表示属性を上書きしません。データベースの設計レベルでテーブルが非表示として設定されている場合、**SET TABLE TITLES**コマンドへの呼び出しにそのテーブルを指定しても、アプリケーションモードにおいてテーブルは表示されません。
- プラグインは常にコマンドによって指定された仮想ストラクチャへアクセスします。

例題

各国で販売する予定の4Dアプリケーションを構築しているとします。この場合、ローカライズの問題を慎重に考慮する必要があります。アプリケーションモードで表示される標準の4Dダイアログボックスとダイナミックな名前を用いたフォームに注意すれば、[Translations] テーブルといくつかのプロジェクトメソッドを使用して、必要なだけ各国向けにローカライズされたフィールドを作成し、使用することによって、ローカライズのニーズに対応できます。

データベースに以下のテーブルを追加します。

Translations
Actual Name
A
Language
A
Translated Name
A

次に、以下に示した TRANSLATE TABLES AND FIELDS プロジェクトメソッドを作成します。このメソッドはデータベースの実際のストラクチャをブラウズし、引数として渡される言語に対応するローカライズ版の作成に必要なすべての *[Translations]* レコードを作成します。

```
  ` TRANSLATE TABLES AND FIELDS プロジェクトメソッド
  ` TRANSLATE TABLES AND FIELDS (Text)
  ` TRANSLATE TABLES AND FIELDS (LanguageCode)

C_TEXT($1)  `ランゲージコード
C_LONGINT($vlTable;$vlField)
C_TEXT($Language)
$Language:=$1

For($vlTable;1;Get last table number)  `各テーブルを渡す
   If($vlTable#((->[Translations])))  `翻訳テーブルを翻訳しない
  `特定ランゲージ用のテーブル名の翻訳があるかどうかをチェックする
      QUERY([Translations];[Translations]LanguageCode=$Language;*)  `目的のランゲージ
      QUERY([Translations]; & ;[Translations]TableID=$vlTable;*)  `テーブル番号
      QUERY([Translations]; & ;[Translations]FieldID=0)  `フィールド番号 = 0はテーブル名
      If(Is table number valid($vlTable))  `テーブルがまだ存在しているかチェックする
         If(Records in selection([Translations])=0)
  `なければ、レコードを作成
            CREATE RECORD([Translations])
            [Translations]LanguageCode:=$Language
            [Translations]TableID:=$vlTable
            [Translations]FieldID:=0
  `翻訳されたテーブル名を入力する必要がある
            [Translations]Translation:=Table name($vlTable)+" in "+$Language
            SAVE RECORD([Translations])
         End if

         For($vlField;1;Get last field number($vlTable))
  `特定ランゲージ用のフィールド名の翻訳があるかどうかをチェックする
            QUERY([Translations];[Translations]LanguageCode=$Language;*)  `目的のランゲージ
            QUERY([Translations]; & ;[Translations]TableID=$vlTable;*)  `テーブル番号
            QUERY([Translations]; & ;[Translations]FieldID=$vlField)  `フィールド番号
            If(Is field number valid($vlTable;$vlField))
               If(Records in selection([Translations])=0)
  `なければ、レコードを作成
                  CREATE RECORD([Translations])
                  [Translations]LanguageCode:=$Language
                  [Translations]TableID:=$vlTable
                  [Translations]FieldID:=$vlField
  `翻訳されたフィールド名を入力する必要がある
                  [Translations]Translation:=Field name($vlTable;$vlField)+" in "+$Language
                  SAVE RECORD([Translations])
               End if
            Else
               If(Records in selection([Translations])#0)
  `フィールドがもはや存在しない場合、翻訳を削除する
                  DELETE RECORD([Translations])
               End if
            End if
         End for
      Else
         If(Records in selection([Translations])#0)
  `テーブルがもはや存在しない場合、翻訳を削除する
            DELETE RECORD([Translations])
         End if
      End if
   End if
End for
```

この時点で以下の行を実行すれば、テーブルタイトルとフィールドタイトルのスペイン語のローカライズ版に必要なすべてのレコードが作成されます。

```
TRANSLATE TABLES AND FIELDS("Spanish")
```

この呼び出しの実行後、新しく作成されたレコードのそれぞれに対して [Translations]Translated Name を入力できます。

最後に、スペイン語のローカライズ版を使用して、データベースの標準な4Dダイアログボックスまたはダイナミックなタイトル付きのフォーム表示します。その際、LOCALIZED TABLES AND FIELDS プロジェクトメソッドを使用して、以下の行を実行します。

```
LOCALIZED TABLES AND FIELDS("Spanish")
  ` LOCALIZED TABLES AND FIELDS global method
  ` LOCALIZED TABLES AND FIELDS (Text)
  ` LOCALIZED TABLES AND FIELDS (LanguageCode)

C_TEXT($1)  `ランゲージコード
C_LONGINT($vlTable;$vlField)
C_TEXT($Language)
C_LONGINT($vlTableNum;$vlFieldNum)
$Language:=$1

  `テーブル名を更新する
ARRAY TEXT($asNames;0)  `SET TABLE TITLES と SET FIELD TITLESを初期化する
ARRAY INTEGER($aiNumbers;0)
QUERY([Translations];[Translations]LanguageCode=$Language;*)
QUERY([Translations]; & ;[Translations]FieldID=0)  `テーブル名
SELECTION TO ARRAY([Translations]Translation;$asNames;[Translations]TableID;$aiNumbers)
SET TABLE TITLES($asNames;$aiNumbers)

  `フィールド名を更新する
$vlTableNum:=Get last table number  `データベース上のテーブルの数を数える
For($vlTable;1;$vlTableNum)  `テーブルを渡す
   If(Is table number valid($vlTable))
      QUERY([Translations];[Translations]LanguageCode=$Language;*)
      QUERY([Translations]; & ;[Translations]TableID=$vlTable;*)
      QUERY([Translations]; & ;[Translations]FieldID#0)  `テーブル名として機能するゼロを避ける
      SELECTION TO ARRAY([Translations]Translation;$asNames;[Translations]FieldID;$aiNumbers)
      SET FIELD TITLES(Table($vlTable)->;$asNames;$aiNumbers)
   End if
End for
```

コードを修正または再編集せずに、新しいローカライズ版をデータベースに追加することができます。

Shift down

Shift down -> 戻り値

引数	型		説明
戻り値	ブール	↩	Shift キーのステータス

説明

Shift down コマンドはShift キーが押されているとTrueを返します。

例題

ボタン bAnyButton に対する以下のオブジェクトメソッドは、ボタンがクリックされた際にどのモディファイアが押されているかによって異なる動作をします。

```
  ` bAnyButton オブジェクトメソッド
 Case of
  `このほかの複数のキーの組み合わせをここでテストすることも可能
  ` ...
    :(Shift down&Windows Ctrl down)
  ` Shift および Windows Ctrl (または Macintosh command) キーが押されてた場合
      DO ACTION1
  ` ...
    :(Shift down)
  `Shift キーだけが押されてた場合
      DO ACTION2
  ` ...
    :(Windows Ctrl down)
  `Windows Ctrl (または Macintosh command) キーだけが押された場合
      DO ACTION3
  ` ...
  `このほかの個々のキーをここでテストすることも可能
  ` ...
 End case
```

SHOW MENU BAR

SHOW MENU BAR
このコマンドは引数を必要としません

説明

SHOW MENU BARコマンドは、メニューバーを表示します。

既にメニューバーが表示されている場合は、このコマンドは何も行いません。

例題

HIDE MENU BARの例を参照してください。

Tool bar height

Tool bar height -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	ツールバーの高さ(ピクセル単位で表示) または、ツールバーが非表示の場合、0

説明

Tool bar heightコマンドは、4Dのツールバーの高さをピクセル単位で返します。ツールバーが非表示の場合、コマンドは0を返します。

Windows Alt down

Windows Alt down -> 戻り値

引数	型		説明
戻り値	ブール	⮐	WindowsのAltキーのステータス (Macintoshの場合は、optionキー)

説明

Windows Alt down はWindowsのAlt キーが押されているとTrueを返します。

Note: Macintoshのプラットフォーム上で呼び出された場合は、Macintoshのoptionキーが押されていると、Windows Alt down はTrueを返します。

例題

Shift down コマンドの例を参照してください。

Windows Ctrl down

Windows Ctrl down -> 戻り値

引数	型	説明
戻り値	ブール	↩ WindowsのCtrlキーのステータス (Macintoshの場合は、commandキー)

説明

Windows Ctrl down はWindowsのCtrlキーが押されているとTrueを返します。

Note: Macintoshのプラットフォーム上で呼び出された場合は、Macintoshのcommandキーが押されていると、Windows Ctrl down はTrueを返します。

例題

Shift down コマンドの例を参照してください。

Get platform interface

Get platform interface -> 戻り値

引数	型		説明
戻り値	倍長整数	←	現在使用中のプラットフォームインタフェース

互換性メモ

このコマンドは互換性に関する理由により維持されています。新しいデータベース (バージョン2004以降の4Dで作成された) では、プラットフォームインタフェースはプログラムによって自動的に管理され、このコマンドは無視されます。

警告: 次のメジャーバージョンでは、プラットフォームインターフェースの設定は動作しなくなります。

HIDE TOOL BAR

HIDE TOOL BAR
このコマンドは引数を必要としません

説明

HIDE TOOL BAR コマンドは4D v14以降、何の効果も持ちません。

SET PLATFORM INTERFACE

SET PLATFORM INTERFACE (interface)

引数	型		説明
interface	倍長整数	→	新しいプラットフォームインタフェース設定 -1 自動 0 Mac OS 7 1 Windows 3.11、NT 3.51 2 Windows 9x 3 Mac OS 9 4 Mac Theme

互換性に関するメモ

このコマンドは互換性のために管理されています。新しいデータベース (バージョン2004以降の4Dで作成されたデータベース)では、プラットフォームインタフェースはプログラムにより自動で管理され、コマンドは動作しま せん。変換されたデータベースではこのコマンドをOn Startupデータベースメソッドで呼び出して、自動インターフェースを全体に適用できます (例題参照)。
警告: 次のメジャーバージョンでは、プラットフォームインターフェースの設定は動作しなくなります。

例題

変換されたデータベースで自動インターフェースプロパティが設定されていない場合、このコマンドを使用して自動プラットフォームインターフェースを適用できます。これを行うには、SET PLATFORM INTERFACEコマンドをOn Startupデータベースメソッドで呼び出します:

```
SET PLATFORM INTERFACE(Automatic Platform)
```

SHOW TOOL BAR

SHOW TOOL BAR
このコマンドは引数を必要としません

説明

SHOW TOOL BAR コマンドは、4D v14以降、何の効果も持ちません。

ユーザフォーム

- ユーザフォームの概要
- CREATE USER FORM
- DELETE USER FORM
- EDIT FORM
- LIST USER FORMS

ユーザフォームの概要

4Dのバージョン2004から、デベロッパはカスタマイズされたフォームを作成したり修正したりする機能を、ユーザに提供することができます。これらの "ユーザフォーム" はその他の4Dのフォームのように使用することが可能です。

概要

ユーザフォームはデベロッパによって作成された標準的な4Dフォーム ("ソース" または"デベロッパ" フォームと呼びます) をベースとして、フォームエディタで**ユーザによる編集可**プロパティが適用されているフォームです。(EDIT FORMコマンドで呼び出される) 簡略化されたフォームエディタを用いて、ユーザはフォームの外観を修正したり、(定義済みオブジェクトのライブラリを使用して) グラフィックオブジェクトを追加したり、エレメントを隠したりするなどの編集ができます。デベロッパは許可するアクションをコントロールできます。

ユーザフォームは2つの異なる方法で使用可能 です:

- ユーザはEDIT FORM コマンドを用いて "ソース" フォームを修正し、独自のニーズへ適応させます。ユーザフォームはローカルに保持され、オリジナルフォームの代わりに自動的に使用されます。

  この動作はデベロッパが設定したダイアログボックスのパラメタに対し、例えば顧客のサイトでフォームに会社のロゴを追加したり、不要なフィールドを隠したりするなどのニーズにこたえるものです。

- "ソース" ファイルは基本テンプレートとして機能します。ユーザはCREATE USER FORMコマンドを使用してこのテンプレートを自由に複製したり、必要なだけコピーを作成したりすることができます。EDIT FORMコマンドを用いてそれぞれのコピーを自由に設定 (内容、名前など) することも可能です。しかし、それぞれのユーザフォームの名前はユニークでなければなりません。そしてFORM SET INPUTコマンドとFORM SET OUTPUTコマンドを用いて、各プロセスで使用するユーザフォームを指定します。

  この動作によりデベロッパはユーザに対してカスタマイズされたレポートなどを作成することができます。

ユーザフォームの格納と管理

ユーザフォームのメカニズムは、4Dのローカルモード、4Dサーバおよび4Dデスクトップにおいて、コンパイルされたデータベースとインタープリタデータベースの両方で機能します。クライアント/サーバモードでは、修正されたユーザフォームはすべてのマシンで利用可能です。

4Dはフォームに加えられた変更を自動的に処理します。フォームが**ユーザによる編集可**として設定されると、デザインモードではフォームがロックされます。フォームオブジェクトへのアクセスを可能にするために、デベロッパはロック解除するアイコンを必ずクリックしなければなりません。この作業により関連するユーザフォームは無効となり、再作成しなければなりません。"ソース" フォームが削除されると、関連するユーザフォームも同じく削除されます。

ユーザフォームは、メインストラクチャファイル (.4DB/.4DC) と同階層に、.4DA 拡張子の別ファイルとして格納されます。このファイルは "ユーザストラクチャファイル" と呼ばれます。このファイルの動作は透過的です。ユーザフォームが存在すると、4Dはそれを使用します (LIST USER FORMS コマンドにより、いつでも有効なユーザフォームを見つけることが可能です) 。またFORM SET INPUTコマンドとFORM SET OUTPUTコマンドはこのファイルの中でユーザフォームを探します。ユーザフォームが無効な場合、そのフォームは削除され、4Dはソースフォームをデフォルトで使用します。
クライアント/サーバでは、メインストラクチャファイルと同じルールに従って、.4DA ファイルがクライアントマシンに配布されます。
この原理により、デベロッパによってストラクチャファイルが更新された場合でも、ユーザフォームを利用可能な状態に維持することができます。

エラーコード

ユーザフォーム管理コマンドを使用する際、特定なエラーコードが返される場合があります。-9750 から -9759までのコードについてはデータベースエンジンエラー (-10600 -> 4004)に記述されています。

ユーザフォームとプロジェクトフォーム

ユーザフォームメカニズムはプロジェクトフォームとの互換性はありません。従って、"ユーザフォーム" テーマのコマンドをプロジェクトフォームに対して使用することはできません。

CREATE USER FORM

CREATE USER FORM (aTable ; form ; userForm)

引数	型		説明
aTable	テーブル	⇒	ソースフォームテーブル
form	文字	⇒	ソースフォームテーブルの名前
userForm	文字	⇒	新規ユーザフォームの名前

説明

CREATE USER FORM コマンドは、引数として渡したテーブルと名前を持つ4Dのテーブルフォームを複製し、userForm という名前の新規ユーザフォームを作成します。

いったん作成されたフォーム userForm は、EDIT FORM コマンドを用いて修正することができます。このコマンドを使用して、単一のソースフォームから X ユーザフォーム (例えば、さまざまなレポートフォーム) を作成することが可能です。

システム変数およびセット

処理が正しく実行されるとOKシステム変数に1が、そうでなければ0が設定されます。

エラー管理

以下の場合においては、エラーが生成されます。

- フォームは既にユーザフォームである。
- userForm の名前は、既存ユーザフォームまたはソースフォームの名前と同じである。
- ユーザが適切なアクセス権を持っていないためフォームにアクセスできない。

ON ERR CALL コマンドによってインストールされたエラー処理メソッドでエラーを検知することができます。

DELETE USER FORM

DELETE USER FORM (aTable ; form ; userForm)

引数	型		説明
aTable	テーブル	→	ユーザフォームテーブル
form	文字	→	ソーステーブルフォームの名前
userForm	文字	→	ユーザフォームの名前

説明

DELETE USER FORM コマンドを使用して、引数 aTable、form と userForm を用いて設定されているユーザフォームを削除できます。

EDIT FORM コマンドでユーザフォームが直接作成された場合、空の文字列 ("") を userForm に渡します。

システム変数およびセット

ユーザフォームが正しく削除されるとOKシステム変数は1に設定されます。そうでなければ0に設定されます。

エラー管理

以下の場合においてはエラーが生成されます。

- ユーザフォームが存在しない。
- ユーザフォームを取り除くための適切なアクセスがユーザにない。

ON ERR CALL コマンドによってインストールされたエラー処理メソッドでエラーを検知することができます。

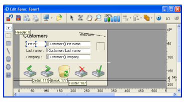
EDIT FORM

EDIT FORM (aTable ; form {; userForm {; library}})

引数	型	説明
aTable	テーブル	→ 修正するフォームのテーブル
form	文字	→ 修正するテーブルフォームの名前
userForm	文字	→ 修正するユーザフォームの名前
library	文字	→ 使用可能なオブジェクトライブラリの完全なパス名

説明

EDIT FORM コマンドを使用して、ユーザフォームエディタで aTable、form およびオプションの引数 userForm を用いて指定されたテーブルフォームを開きます。


Edit Form: Form1
Header: 0
Customers
First n
[Customers]First name
Last name :
[Customers]Last name
Company :
[Customers]Company
Detail: 177
Break: 177
Footer: 187
100%
1/1
0
50
100
150
200
250
300
350
400
450
500
50
100
150
200

Note: エディタのウィンドウはプロセスでの最初のウィンドウの場合のみ開きます。つまり、通常では、エディタを表示するために新しいプロセスを開くことが必要となります。

userFormに空の文字列を渡し、formとリンクされているユーザフォームがまだない場合、ソースフォームがエディタに表示されます。そして修正されたフォームがユーザストラクチャファイル (.4DA) にコピーされ代替 form として使用されます。

このコマンドを用いて、ユーザフォームが既に form から作成されていた場合、そのユーザフォームがエディタに表示されます。ソースフォームから開始をしたい場合は、DELETE USER FORM コマンドを用いてユーザフォームを必ず 最初に削除しなければなりません。

引数 userForm を用いて、修正するユーザフォーム (CREATE USER FORM コマンドを用いて作成された) を設定することができます。この場合、そのフォームがエディタに表示されます。

オプションの引数 library には、ユーザがフォームをカスタマイズするために使用可能なオブジェクトライブラリの完全なアクセスパスを渡します。ユーザフォームエディタと一緒に使用すると、オブジェクトライブラリから、グラフィックプロパティや自動アクションを保持したままオブジェクトをペーストできます。メソッド付きのオブジェクトはライブラリに表示されません。ライブラリオブジェクトの名前、変数、タイプがユーザフォーム (とそのオブジェクト) と互換性があるかを確認するのはデベロッパの責任です。
クライアント/サーバモードでは、ライブラリは**Plugins** フォルダと同じレベルにあるデータベースの**Resources** フォルダに存在していなければなりません。そうであればすべてのクライアントマシンで利用することができます。ライブラリが有効な場合、そのライブラリはフォームウィンドウとともに開きます。
オブジェクトライブラリに関する詳細については、4DドキュメントのDesign Reference マニュアルを参照してください。

例題

この例題では、ユーザはライブラリを選択して、ダイアログフォームを修正します。

```
MAP FILE TYPES("4DLB";"4IL";"4D Library")
$vALib:=Select document(1;"4DLB";"Please select a library";0)
If(OK=1)
 `ライブラリが選択されました。
   $vALibPath:=Document
Else
   $vALibPath:=""
End if

EDIT FORM([Dialogs];"Welcome";"Lib_Logos.4il")
If(OK=1)
 `修正されたフォームの表示。
   DIALOG([Dialogs];"Welcome")
End if
```

システム変数およびセット

フォームに対して行われた変更が保存されるとOKシステム変数に1が設定されます。そうでなければ0に設定されます。

エラー管理

以下の場合、エラーが生成されます。

- デザインモードで、フォームがユーザによって編集可能と指定されていない場合や、フォームが存在しない場合。
- フォームが既に開いていて、他のプロセスで修正されている場合。
- ユーザが適切なアクセス権を持っていないため、フォームにアクセスできない場合。

ON ERR CALL コマンドを用いてインストールされたエラー処理メソッドでこのエラーを検知することができます。

LIST USER FORMS

LIST USER FORMS (aTable ; form ; userFormArray)

引数	型	説明
aTable	テーブル	→ ソースフォームテーブル
form	文字	→ ソーステーブルフォームの名前
userFormArray	文字配列	← ソースフォームに属する ユーザフォームの名前

説明

LIST USER FORMS コマンドは、引数 table と form に設定されている開発者フォーム (テーブルフォーム) に属するユーザフォームの名前を配列 userFormArray に代入します。

EDIT FORM コマンドを使用して、ユーザフォームが直接作成された場合、userFormArray が格納する唯一のアイテムは空の文字列 ("") です。

指定された開発者フォームに対してユーザフォームがない場合、配列は空です。

ランゲージ

- Command name
- Count parameters
- Current method name
- EXECUTE METHOD
- EXECUTE METHOD IN SUBFORM
- Get pointer
- Is a variable
- Nil
- NO TRACE
- RESOLVE POINTER
- Self
- TRACE
- Type

Command name

Command name (command) -> 戻り値

引数	型		説明
command	倍長整数	→	コマンド番号
戻り値	文字	↩	ローカライズされたコマンド名

説明

Command name コマンドは、引数commandに渡したコマンド番号のコマンド名を返します。

4Dは、メソッドで使用するキーワード、定数、およびコマンド名の動的な翻訳を統合しています。4Dの日本語版を使用している場合、次のように記述します:

```
DEFAULT TABLE([MyTable])
ALL RECORDS([MyTable])
```

同じコードがフランス語版の4Dでは以下のようになります:

```
TABLE PAR DEFAUT([MyTable])
TOUT SELECTIONNER([MyTable])
```

ただし、4DにはユニークなEXECUTE FORMULAコマンドがあり、これを使用すると、データベースがコンパイルされていてもただちにコードを作成して、このコードを実行することができます。

日本語版でEXECUTE FORMULA文を使用して作成されたコードの例は、以下のようになります:

```
EXECUTE FORMULA("DEFAULT TABLE([MyTable])")
EXECUTE FORMULA("ALL RECORDS([MyTable])")
```

同じコードがフランス語版の4Dで開かれると以下のようになります:

```
EXECUTER FORMULE("DEFAULT TABLE([MyTable])")
EXECUTER FORMULE("ALL RECORDS([MyTable])")
```

4Dは、EXECUTE FORMULA（日本語版）をEXECUTER FORMULE（フランス語版）に自動で翻訳しますが、コマンドに渡したテキスト文を翻訳することはできません。

アプリケーションでEXECUTE FORMULA コマンドを使用する場合には、Command nameを使用して異なる言語間のローカライズにおける実行文の問題を解決し、文を言語に依存しないようにすることができます。コードの例は、以下のようになります:

```
EXECUTE FORMULA(Command name(46)+"([MyTable])")
EXECUTE FORMULA(Command name(47)+"([MyTable])")
```

4Dのフランス語版を使用すると、このコードは以下のようになります:

```
EXECUTER FORMULE(Nom commande(46)+"([MyTable])")
EXECUTER FORMULE(Nom commande(47)+"([MyTable])")
```

Note: コマンド毎のコマンド番号は、コマンドドキュメントのプロパティエリアに書かれています。

例題 1

データベースのすべてのテーブルについて、各テーブルの標準データ入力に使用するための"INPUT FORM"というフォームがあります。そこで、ポインタまたはテーブル名を渡すテーブルのカレント入力フォームとしてこのフォームを設定する、汎用的なプロジェクトメソッドを追加する場合には、次のように記述します:

```
  ` STANDARD INPUT FORM project method
  ` STANDARD INPUT FORM ( ポインター {; 文字 })
  ` STANDARD INPUT FORM ( ->Table {; TableName })
C_POINTER($1)
C_STRING(31;$2)

If(Count parameters>=2)
   EXECUTE FORMULA(Command name(55)+"(["+$2+"];"+Char(Double quote)+"INPUT FORM"+Char(Double quote)+")")
Else
   If(Count parameters>=1)
      FORM SET INPUT($1->;"INPUT FORM")
   End if
End if
```

このプロジェクトメソッドがデータベースに追加された後、以下のように記述します:

```
STANDARD INPUT FORM(->[Employees])
STANDARD INPUT FORM("Employees")
```

Note: 通常、汎用ルーチンを記述する場合には、ポインタを使用することをお勧めします。その理由としては、まず、データベースがコンパイルされているとき、そのコードもコンパイルモードで実行されるからです。次に前述の例でもあるようにテーブルの名前を変更すると、コードは正しく動作しなくなるためです。ただし、EXECUTE FORMULAコマンドを使用すれば問題が解決する場合もあります。

例題 2

フォームで、一般的なサマリーレポートコマンドのドロップダウンリストを作成します。ドロップダウンリストのオブジェクトメソッドに、次のように記述します:

```
Case of
   :(Form event=On Before)
      ARRAY TEXT(asCommand;4)
      asCommand{1}:=Command name(1) ` Sum
      asCommand{2}:=Command name(2) ` Average
      asCommand{3}:=Command name(4) ` Min
      asCommand{4}:=Command name(3) ` Max
  ` ...
End case
```

4Dの日本語版ではドロップダウンリストに、Sum、Average、Min、Maxが表示されます。フランス語版では、ドロップダウンリストには、Somme、Moyenne、Min、Maxが表示されます。

Count parameters

Count parameters -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	実際に渡された引数の数

説明

Count parameters コマンドは、プロジェクトメソッドに渡された引数の数を返します。

警告: Count parametersは、他のメソッド(プロジェクトメソッド、その他)から呼び出されるプロジェクトメソッド内でのみ意味を持ちます。Count parametersを呼び出すプロジェクトメソッドがメニューに割り当てられている場合、Count parametersは0を返します。

例題 1

4Dプロジェクトメソッドは右側から始まるオプションの引数を受け付けます。
例えば、メソッドMyMethod(a;b;c;d)は以下のように呼び出すことができます:

```
MyMethod(a;b;c;d) ` すべての引数を渡す
MyMethod(a;b;c) ` 最後の引数を省略
MyMethod(a;b) ` 後ろ2つの引数を省略
MyMethod(a) ` 1番目の引数の実を渡す
MyMethod ` 引数を渡さない
```

MyMethod内でCount parametersを使用し、実際の引数の数を取得してその数に応じて異なる処理を実行することができます。以下の例では、テストメッセージを表示し、4D Writeエリアにテキストを挿入、またはディスク上のドキュメントにテキストを送信しています:

```
  ` APPEND TEXT Project Method
  ` APPEND TEXT ( Text { ; Long { ; Time } } )
  ` APPEND TEXT ( Text { ; 4D Write Area { ; DocRef } } )

 C_TEXT($1)
 C_TIME($2)
 C_LONGINT($3)

 MESSAGE($1)
 If(Count parameters>=3)
    SEND PACKET($3;$1)
 Else
    If(Count parameters>=2)
       WR INSERT TEXT($2;$1)
    End if
 End if
```

このプロジェクトメソッドをアプリケーションに追加すると以下のように記述できます:

```
APPEND TEXT(vtSomeText) ` テキストメッセージの表示のみ
APPEND TEXT(vtSomeText;$wrArea) ` テキストメッセージを表示して、$wrAreaエリアに追加
APPEND TEXT(vtSomeText;0;$vhDocRef) ` テキストメッセージを表示して$vhDocRefに書き込む
```

例題 2

4Dのプロジェクトメソッドは、右側から始めて、可変数の同タイプの引数を受け付けます。これらの引数を宣言するには、コンパイル命令を使用し、変数として${N}を渡します。Nは最初の引数を示します。Count parametersを使い、Forループと引数の間接参照構文を用いてこれらの引数にアクセスすることができます。この例は関数で、引数として受け取った最も大きな数値を返します:

```
  ` Max of Project Method
  ` Max of ( Real { ; Real2... ; RealN } ) -> Real
  ` Max of ( Value { ; Value2... ; ValueN } ) -> Greatest value

 C_REAL($0;${1}) ` 全ての引数および戻り値の型は実数
 $0:=${1}
 For($vlParam;2;Count parameters)
    If(${$vlParam}>$0)
       $0:=${$vlParam}
    End if
 End for
```

このプロジェクトメソッドをアプリケーションに追加すると以下のように記述できます:

```
vrResult:=Max of(Records in set("Operation A");Records in set("Operation B"))
```

または:

```
vrResult:=Max of(r1;r2;r3;r4;r5;r6)
```

Current method name

Current method name -> 戻り値

引数	型		説明
戻り値	文字	←	呼び出しメソッド名

説明

Current method nameコマンドは、このコマンドを呼び出したメソッド名を返します。このコマンドは汎用メソッドをデバッグするときに有効です

呼び出しメソッドのタイプにより、返される文字列は次のようになります:

呼び出しメソッド	**返される文字列**
データベースメソッド	メソッド名
トリガ	トリガ -> [テーブル名]
プロジェクトメソッド	メソッド名
テーブルフォームメソッド	[テーブル名].フォーム名
プロジェクトフォームメソッド	フォーム名
テーブルフォームオブジェクトメソッド	[テーブル名].フォーム名.オブジェクト名
プロジェクトフォームオブジェクトメソッド	フォーム名.オブジェクト名
コンポーネントプロジェクトメソッド	メソッド名
コンポーネントプロジェクトフォームメソッド	フォーム名(コンポーネント名)
コンポーネントプロジェクトフォームオブジェクトメソッド	フォーム名(コンポーネント名).オブジェクト名(コンポーネント名)

このコマンドを4Dフォーミュラ内で呼び出すことはできません。

注: コンパイルモード中でこのコマンドを使用するために、データベース設定のコンパイラーページで**範囲チェック**オプションを選択しなければなりません。
メソッド(あるいはメソッドの一部)を局所的に範囲チェック無効にしたい場合は、以下の特別なコメントを使用することができます:

- `` `%R- `` 範囲チェックを無効にする
- `` `%R+ `` 範囲チェックを有効にする
- `` `%R* `` 環境設定で設定された範囲チェックの初期設定を使用する

EXECUTE METHOD

EXECUTE METHOD (methodName {; result | * {; param}}{; param2 ; ... ; paramN})

引数	型		説明
methodName	文字	➡	実行するプロジェクトメソッド名
result | *	変数, 演算子	⬅	メソッドの結果を受け取る変数 または *: メソッドが結果を返さない場合
param	式	➡	メソッドの引数

説明

EXECUTE METHOD コマンドは、param1...paramNを引数に渡して、methodNameプロジェクトメソッドを実行します。データベースまたはコマンドを実行するコンポーネントから呼び出し可能なメソッド名を渡すことができます。

resultには、methodName の実行結果を受け取る変数を渡します (methodName内で$0に置かれる値)。メソッドが値を返さない場合、* を2番目の引数に渡します。メソッドが結果を返さず、引数を必要としない場合、methodName 引数のみを渡します。

実行コンテキストは呼び出しメソッドです。すなわちカレントフォームやカレントフォームイベントは呼び出されるメソッド内でも有効です。

このコマンドをmethodName にホストデータベースのメソッドを渡してコンポーネントから呼び出す場合 (あるいはその逆の場合)、メソッドはメソッドプロパティ内でホストデータベースとコンポーネントで共有されるよう設定しなければなりません。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

EXECUTE METHOD IN SUBFORM

EXECUTE METHOD IN SUBFORM (subformObject ; methodName {; return {; param} {; param2 ; ... ; paramN}})

引数	型		説明
subformObject	テキスト	→	サブフォームオブジェクトの名称
methodName	テキスト	→	実行するプロジェクトメソッドの名前
return	演算子, 変数	→	メソッドが値を戻さない場合 *
		←	メソッドから返される値
param	変数	→	メソッドに渡す引数

説明

EXECUTE METHOD IN SUBFORM コマンドを使用して サブフォームオブジェクトsubformObject のコンテキストでプロジェクトメソッドmethodName を実行できます。

呼ばれるプロジェクトメソッドは任意の数の引数をparam に受け取ることができ、また戻り値はreturnに返されます。メソッドが値を戻さない場合は、returnに * を渡します。

methodNameにはデータベースあるいはコマンドを実行するコンポーネントからアクセスが可能なプロジェクトメソッドを指定できます。実行コンテキストは呼び出されたメソッド内でも保持されます。つまりカレントフォームおよびカレントフォームイベントは指定されたまま引き継がれます。サブフォームがコンポーネント由来の場合、メソッドはコンポーネントに属していなければならず、また"コンポーネントとホストデータベースで共有する"プロパティがチェックされていなければなりません。

このコマンドは (subformObject オブジェクトを含む) 親フォームのフォームメソッドから実行しなくてはなりません。

注: methodName メソッドは、subformObject がカレントページに見つからないか、インスタンス化されていない場合、実行されません。

例題 1

親フォーム"Company"中にサブフォーム"ContactDetail"が置かれています。ContactDetailフォームが設定されたサブフォームオブジェクトの名前は"ContactSubform"です。ここでcompanyのフィールド値に基づき、サブフォーム内の特定の要素のアピアランスを変更したいとします (例えば[Company]City="New York"のときは"contactname"を赤に、[Company]City="San Diego"のときは青にするなど)。このメカニズムはSetToColorメソッドに実装されています。この結果を得るために、SetToColor メソッドをCompany親フォームの"On Load"フォームイベントのプロセスから直接呼び出すことはできません。なぜなら"contactname"オブジェクトはカレントフォームではなく、"ContactSubform"サブフォームオブジェクト中に表示されているフォームに属しているからです。そのため正しく動作させるために、メソッドはEXECUTE METHOD IN SUBFORMコマンドを使用して実行されなければなりません。

```
Case of
   :(Form event=On Load)
      Case of
         :([Company]City="New York")
            $Color:=$Red
         :([Company]City="San Diego")
            $Color:=$Blue
         Else
            $Color:=$Black
      End case
      EXECUTE METHOD IN SUBFORM("ContactSubform";"SetToColor";*;$Color)
End case
```

例題 2

コンポーネントとして使用される予定のデータベースを開発しています。このデータベースには共有プロジェクトフォーム (例として"Calender"と名付けます) が含まれています。またこのフォームにはダイナミック変数やカレンダを調整するための公開プロジェクトメソッド (SetCalendarDate(varDate)) が含まれています。
このメソッドがCalenderフォームメソッドで直接使用される場合、開発者は直接このメソッドをフォームの"On Load"イベントで呼び出すことができます:

```
SetCalendarDate(Current date)
```

しかしホストデータベースのコンテキストで、プロジェクトフォームに2つの"Calendar"サブフォーム"Cal1"および"Cal2"が配置されている状況を想像してください。Cal1の日付を2010/01/01、Cal2の日付を2010/05/05に設定する必要があるとします。このケースでは、コンポーネントメソッドSetCalendarDate を直接呼び出すことはできません。なぜならコンポーネントメソッドはどちらのフォームに処理を適用したらよいかわからないからです。そこで、このメソッドを以下の方法で呼び出す必要があります:

```
EXECUTE METHOD IN SUBFORM("Cal1";"SetCalendarDate";*;!10/01/01!)
EXECUTE METHOD IN SUBFORM("Cal2";"SetCalendarDate";*;!10/05/05!)
```

例題 3

上級例題: 先と同じ状況で、ここでは汎用メソッドを作成してみます:

```
 // SetCalendarDate メソッドの内容
C_DATE($1)
C_TEXT($2)
Case of
   :(Count parameters=1)
 // 標準のメソッド実行 (フォーム内で呼び出された場合)
 // またはケース2からの再帰呼び出し

   :(Count parameters=2)
 // 外部呼出し、
 // 1つの引数で再帰呼び出しを行う
      EXECUTE METHOD IN SUBFORM($2;"SetCalendarDate";*;$1)
End case
```

Get pointer

Get pointer (varName) -> 戻り値

引数	型	説明
varName	文字	→ プロセスまたはインタプロセス変数の名前
戻り値	ポインター	↩ プロセスまたはインタープロセス変数へのポインタ

説明

Get pointer コマンドは、varNameに渡した名前を持つ変数へのポインタを返します。

フィールドへのポインタを取得するにはFieldを使用します。テーブルへのポインタを取得するにはTableを使用します。

Note: Get pointerは、例えば**ArrName+"{3}"**のような式を受け入れます。しかし、二次元配列要素 (**ArrName+"{3}{5}"**) や変数要素参照 (**ArrName+"{myVar}"**) は使用できません。

例題

フォーム上で、5 x 10のグリッドの入力可能な変数を作成し、それぞれv1, v2... v50という名前を付けます。これらの変数をすべて初期化するには次のようにします:

```
   ...
For($vlVar;1;50)
   $vpVar:=Get pointer("v"+String($vlVar))
   $vpVar->:=""
End for
```

Is a variable

Is a variable (aPointer) -> 戻り値

引数	型	説明
aPointer	ポインター	→ テストするポインタ
戻り値	ブール	↩ TRUE = 変数のポインタ FALSE = 変数以外のポインタ

説明

Is a variableコマンドは、aPointerが定義済み変数を参照する場合にはTrueを返します。その他の場合（フィールドやテーブルへのポインター、Nilポインター等）、この関数はFalseを返します。

参照されている変数の名前やフィールド番号を知りたい場合、RESOLVE POINTERコマンドを使用します。

Nil

Nil (aPointer) -> 戻り値

引数	型		説明
aPointer	ポインター	→	テストするポインタ
戻り値	ブール	↩	TRUE = Nil ポインタ (->[]) FALSE = 既存のオブジェクトへの有効なポインタ

説明

Nilコマンドは、aPointerがNilポインタ (->[]) の場合に**True**を返します。その他の場合（フィールドやテーブル、変数へのポインタ）、この関数はFalseを返します。

ポインターの参照先である変数の名前やフィールド番号を知りたい場合、RESOLVE POINTERコマンドを使用します。

NO TRACE

NO TRACE
このコマンドは引数を必要としません

説明

データベースの開発時に、メソッドの実行を検証する際にNO TRACEコマンドを使用します。

NO TRACEコマンドは、TRACEコマンド、エラー、ユーザによって起動されたデバッガを終了します。これはデバッグウィンドウの**トレースなし**ボタンをクリックしたのと同じ動作です。

コンパイルされたデータベースでは、NO TRACEコマンドは無視されます。

RESOLVE POINTER

RESOLVE POINTER (pointer ; varName ; tableNum ; fieldNum)

引数	型		説明
pointer	ポインター	→	参照オブジェクトを取得するポインタ
varName	文字	←	参照された変数の名前または空の文字列
tableNum	倍長整数	←	参照されたテーブルまたは配列要素の番号 または 0 あるいは -1
fieldNum	倍長整数	←	参照されたフィールドの番号 または 0

説明

RESOLVE POINTER コマンドは、pointer式によって参照されるオブジェクトの情報を取得し、引数varName、tableNum、fieldNumに返します。

参照されるオブジェクトの種類によって、RESOLVE POINTERコマンドは、以下のような値を返します:

参照されるオブジェクト	引数		
	varName	**tableNum**	**fieldNum**
なし (NILポインタ)	"" (空の文字列)	0	0
変数	変数名	-1	0
配列	配列名	-1	0
配列要素	配列名	要素番号	0
テーブル	"" (空の文字列)	テーブル番号	0
フィールド	"" (空の文字列)	テーブル番号	フィールド番号

Notes:

- pointerに渡す値がポインタ式でない場合には、シンタックスエラーが発生します。
- RESOLVE POINTER コマンドは、ローカル変数のポインタには使用できません。何故なら、ローカル変数は、同じ名前で異なる場所で定義できるため、目的のローカル変数を特定することができないからです。

例題 1

フォーム内で、v1, v2... v100という名前で入力可能な変数100個を作成します。これを実行するには、以下のような手順を実行します:

a. 入力可能な変数を1つ作成し、vと名付ける。

b. オブジェクトのプロパティを設定する。

c. オブジェクトに以下のメソッドを作成する:

```
DoSomething(Self) ` DoSomething はデータベースのプロジェクトメソッド
```

d. この時点で、必要な回数だけ変数を複製することも、フォームエディタの「グリッドで変数作成」機能を使用することもできる。

e. DoSomethingメソッド内で、このメソッドが呼び出される変数のインデックスを知る必要がある場合には、以下のように記述する:

```
RESOLVE POINTER($1;$vsVarName;$vlTableNum;$vlFieldNum)
$vlVarNum:=Num(Substring($vsVarName;2))
```

この方法でフォームを作成することによって、100個の変数のためのメソッドを一度書くだけで済むことに注目してください。DoSomething (1), DoSomething (2)...,DoSomething (100)を作成する必要はありません。

例題 2

デバッグのために、メソッドへの2番目の引数 ($2) がテーブルへのポインタであることを確認する必要があるとします。この場合、メソッドの最初で、以下のように記述します:

```
 ` ...
If(<>DebugOn)
   RESOLVE POINTER($2;$vsVarName;$vlTableNum;$vlFieldNum)
   If(Not(($vlTableNum>0)&($vlFieldNum=0)&($vsVarName="")))
 ` 警告： ポインタはテーブルへの参照ではない
      TRACE
   End if
End if
 ` ...
```

例題 3

DRAG AND DROP PROPERTIES コマンドの例を参照.

Self

Self -> 戻り値

引数	型		説明
戻り値	ポインター	↩	メソッドが現在実行されているフォーム オブジェクトへのポインタ そうでない場合コンテキストの外側ではNil (->[])

互換性に関する注意

このコマンドは互換性の目的で保持されています。4D v12からはOBJECT Get pointerコマンドの利用が推奨されます。

説明

Self コマンドはオブジェクトメソッドが現在実行されているオブジェクトへのポインタを返します。

Self はオブジェクトメソッド自身内で変数を参照するために使用されます。コマンドはオブジェクトメソッド内で呼ばれるか、オブジェクトメソッドから直接あるいは間接に呼ばれたプロジェクトメソッド内で呼ばれた場合に、有効なポインタを返します。

Self が上記のコンテキスト以外で呼ばれた場合、Nilポインタ (->[]) が返されます。

Tip: Self は、フォーム上の複数のオブジェクトに同じ処理を実行させる場合に便利です。

Note: このコマンドがリストボックスのコンテキストで使用される場合、コマンドはそのコンテキストに応じリストボックスまたはリストボックスの列へのポインタを返します。詳細はの節を参照してください。

例題

RESOLVE POINTER コマンドの例を参照

TRACE

TRACE
このコマンドは引数を必要としません

説明

データベースの開発時に、TRACEコマンドを使用してメソッドをトレースすることができます。

TRACEコマンドは、カレントプロセス用の4Dのを起動します。デバッグウィンドウはコードの次の行が実行される前に表示されます。実行するコードを表示しながらメソッドの実行を続けることができます。コードの実行中に、**Alt+Shift+右クリック** (Windows) 、または**Control+Option+Command+クリック** (Macintosh) でもデバッガを起動することができます。

コンパイルされたデータベースでは、TRACEコマンドは無視されます。

4D Server: ストアドプロシージャのコンテキスト内で実行されたプロジェクトメソッドからTRACEコマンドをコールすると、デバッグウィンドウはサーバマシン上に表示されます。

Tip: On ActivateおよびOn Deactivateイベントが選択されたフォームを使用する場合、TRACEコマンドを使用しないでください。デバッグウィンドウが表示されるたびに、これらのイベントが起動されて、この2つのイベントとデバッグウィンドウとの間で永久ループになってしまいます。このような状況になった場合には、**トレースなし**ボタンを**Shiftキーを押しながらクリック**します。以降、そのプロセス内ではTRACEコマンドが無視されます。

例題

次のコードでは、プロセス変数**BUILD_LANG**に"US"あるいは"FR"という値であることを期待しています。それ以外の値である場合に、プロジェクトメソッドDEBUGを呼び出します:

```
  ` ...
 Case of
    :(BUILD_LANG="US")
       vsBHCmdName:=[Commands]CM US Name
    :(BUILD_LANG="FR")
       vsBHCmdName:=[Commands]CM FR Name
    Else
       DEBUG("Unexpected BUILD_LANG value")
 End case
```

DEBUG プロジェクトメソッドを次に示します:

```
  ` DEBUG Project Method
  ` DEBUG (Text)
  ` DEBUG (Optional Debug Information)

 C_TEXT($1)

 If(<>vbDebugOn)  ` インタープロセス変数はOn Startupメソッドで設定される
    If(Is compiled mode)
       If(Count parameters>=1)
          ALERT($1+Char(13)+"Call Designer at x911")
       End if
    Else
       TRACE
    End if
 End if
```

Type

Type (fieldVar) -> 戻り値

引数	型		説明
fieldVar	フィールド, 変数	→	テストするフィールドまたは変数
戻り値	倍長整数	↩	データタイプ番号

説明

Type コマンドは、fieldVarで渡したフィールドや変数のタイプを示す数値を返します。

4Dは、以下のような定義済み定数を持っています:

定数	型	値
Array 2D	倍長整数	13
Boolean array	倍長整数	22
Date array	倍長整数	17
Integer array	倍長整数	15
Is Alpha Field	倍長整数	0
Is BLOB	倍長整数	30
Is Boolean	倍長整数	6
Is Date	倍長整数	4
Is Float	倍長整数	35
Is Integer	倍長整数	8
Is Integer 64 bits	倍長整数	25
Is JSON null	倍長整数	255
Is LongInt	倍長整数	9
Is object	倍長整数	38
Is Picture	倍長整数	3
Is Pointer	倍長整数	23
Is Real	倍長整数	1
Is String Var	倍長整数	24
Is Subtable	倍長整数	7
Is Text	倍長整数	2
Is Time	倍長整数	11
Is Undefined	倍長整数	5
LongInt array	倍長整数	16
Object array	倍長整数	39
Picture array	倍長整数	19
Pointer array	倍長整数	20
Real array	倍長整数	14
String array	倍長整数	21
Text array	倍長整数	18

Notes:

- Type はグラフ変数に適用されると 9 (Is Longint) を返します。
- 4D v11より、Type が2次元配列の行に適用されると、実際の配列の型を返すようになりました。以前はArray 2D が返されました (例題 4参照)。
- 4D v11より、Type は文字変数や文字配列に対しそれぞれ Is Text や Text array を返すようになりました。このバージョンからは文字変数とテキスト変数の間に違いはありません。

Type をフィールド、インタープロセス変数、プロセス変数、ローカル変数、そしてこれらのオブジェクトタイプを参照するポインタの逆参照に適用することができます。Type を引数 ($1,$2..., ${...}) やプロジェクトメソッド、戻り値 ($0)に使用できます。

例題 1

APPEND DATA TO PASTEBOARD コマンドの例題参照

例題 2

DRAG AND DROP PROPERTIES コマンドの例題参照

例題 3

以下のプロジェクトメソッドは、テーブルのカレントレコードのフィールドの一部あるいはすべてをクリアします。テーブルは、ポインタ引数として渡されます。カレントレコードの削除や変更は行いません

```
  ` EMPTY RECORD Project Method
  ` EMPTY RECORD ( Pointer {; Long } )
  ` EMPTY RECORD ( -> [table] { ; type Flags } )

 C_POINTER($1)
 C_LONGINT($2;$vlTypeFlags)

 If(Count parameters>=2)
    $vlTypeFlags:=$2
 Else
    $vlTypeFlags:=0xFFFFFFFF
 End if
 For($vlField;1;Get last field number($1))
    $vpField:=Field(Table($1);$vlField)
    $vlFieldType:=Type($vpField->)
    If($vlTypeFlags ??$vlFieldType )
       Case of
          :(($vlFieldType=Is Alpha Field)|($vlFieldType=Is Text))
             $vpField->:=""
          :(($vlFieldType=Is Real)|($vlFieldType=Is Integer)|($vlFieldType=Is LongInt))
             $vpField->:=0
          :($vlFieldType=Is Date)
             $vpField->:=!00/00/00!
          :($vlFieldType=Is Time)
             $vpField->:=?00:00:00?
          :($vlFieldType=Is Boolean)
             $vpField->:=False
          :($vlFieldType=Is Picture)
             C_PICTURE($vgEmptyPicture)
             $vpField->:=$vgEmptyPicture
          :($vlFieldType=Is Subtable)
             Repeat
                ALL SUBRECORDS($vpField->)
                DELETE SUBRECORD($vpField->)
             Until(Records in subselection($vpField->)=0)
          :($vlFieldType=Is BLOB)
             SET BLOB SIZE($vpField->;0)
       End case
    End if
 End for
```

このプロジェクトメソッドをデータベースに作成後、以下のように使用できます:

```
  ` テーブル [Things To Do]のカレントレコードを空にする
 EMPTY RECORD(->[Things To Do])

  ` テーブル [Things To Do]のカレントレコードのテキスト, BLOB, ピクチャを空にする
 EMPTY RECORD(->[Things To Do];0?+Is Text?+Is BLOB?+Is Picture)

  ` テーブル [Things To Do] のカレントレコードの文字フィールド以外を空にする
 EMPTY RECORD(->[Things To Do];-1?-Is Alpha Field)
```

例題 4

汎用コードを書くなど特別なケースで、配列が標準の配列か、2次元配列の行かを知りたい場合があるとします。以下のように書くことができます:

```
 ptrmyArr:=->myArr{6} ` myArr{6} 2D 配列か?
 RESOLVE POINTER(ptrmyArr;varName;tableNum;fieldNum)
 If(varName#"")
    $ptr:=Get pointer(varName)
    $thetype:=Type($ptr->)
  ` myArr{6} が2次元配列の行なら$thetype は 13になる
 End if
```

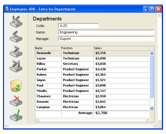
リストボックス

リストボックスオブジェクトの管理
階層リストボックスの管理
LISTBOX COLLAPSE
LISTBOX DELETE COLUMN
LISTBOX DELETE ROWS
LISTBOX DUPLICATE COLUMN New 14.0
LISTBOX EXPAND
LISTBOX Get array New 14.0
LISTBOX GET ARRAYS
LISTBOX GET CELL POSITION
LISTBOX Get column formula
LISTBOX Get column width
LISTBOX Get footer calculation
LISTBOX Get footers height
LISTBOX GET GRID
LISTBOX GET GRID COLORS
LISTBOX Get headers height
LISTBOX GET HIERARCHY
LISTBOX Get information
LISTBOX Get locked columns
LISTBOX Get number of columns
LISTBOX Get number of rows
LISTBOX GET OBJECTS New 14.0
LISTBOX GET PRINT INFORMATION
LISTBOX Get row color New 14.0
LISTBOX Get row font style New 14.0
LISTBOX Get rows height
LISTBOX Get static columns
LISTBOX GET TABLE SOURCE
LISTBOX INSERT COLUMN
LISTBOX INSERT COLUMN FORMULA
LISTBOX INSERT ROWS
LISTBOX MOVE COLUMN New 14.0
LISTBOX MOVED COLUMN NUMBER
LISTBOX MOVED ROW NUMBER
LISTBOX SELECT BREAK
LISTBOX SELECT ROW
LISTBOX SET ARRAY New 14.0
LISTBOX SET COLUMN FORMULA
LISTBOX SET COLUMN WIDTH
LISTBOX SET FOOTER CALCULATION
LISTBOX SET FOOTERS HEIGHT
LISTBOX SET GRID
LISTBOX SET GRID COLOR
LISTBOX SET HEADERS HEIGHT
LISTBOX SET HIERARCHY
LISTBOX SET LOCKED COLUMNS
LISTBOX SET ROW COLOR New 14.0
LISTBOX SET ROW FONT STYLE New 14.0
LISTBOX SET ROWS HEIGHT
LISTBOX SET STATIC COLUMNS
LISTBOX SET TABLE SOURCE
LISTBOX SORT COLUMNS

リストボックスオブジェクトの管理

このテーマのコマンドは、リストボックス型のフォームオブジェクトを扱うために設けられました。

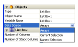
リストボックスを使用するとデータを列と選択可能な行の形式で表現できます。さらに値の入力や列のソート、階層表示、行ごとの色の変更などさらに数多くの機能が用意されています。



4Dのフォームエディタでリストボックスオブジェクトタイプを完全に設定することが可能で、またプログラムから管理することもできます。フォームエディタでのリストボックスタイプのオブジェクトの作成および設定に関する詳細はデザインリファレンスマニュアルを参照してください。
リストボックスオブジェクトのプログラミングは、4Dの他のリストフォームオブジェクトと同じ方法で行われます。ただし以下の節で説明するように、特定のルールに従わなくてはなりません。

データソースおよび値の管理に関する原則

リストボックスオブジェクトには1つ以上の列を含めることができます。それぞれの列には4D配列またはレコードのセレクションを関連付けることができます。レコードセレクションの場合、それぞれの列にはフィールドまたは式を関連付けます。
1つのリストボックス内に配列とセレクション両方をデータソースとして指定することはできません。フォームエディタ上でリストボックスを作成する際に、プロパティリストでデータソースを設定します。それをプログラムで変更することはできません。



配列タイプのリストボックス

このタイプのリストボックスでは、それぞれの列に4Dの1次元配列を割り当てなければなりません。ポインター配列を除きすべてのタイプの配列を使用できます。フォームエディターやOBJECT SET FORMATコマンドを使用して、列ごとに表示フォーマットを指定できます。

リストボックスのハイレベルコマンド (LISTBOX INSERT ROWSやLISTBOX DELETE ROWS等) や配列操作コマンドを使用して、列の値 (データ入力や表示) を管理します。
例えば列の内容を初期化するには、以下の命令を使用できます:

```
ARRAY TEXT(ColumnName;size)
```

リストを使用することもできます:

```
LIST TO ARRAY("ListName";ColumnName)
```

警告: リストボックスが異なる配列サイズの列を含むとき、もっとも小さい配列サイズの数だけを表示します。開発者は、各配列の要素数を同じになるようにすべきで す。一つでも、リストボックスの列が空(配列未定義だったり、正しく再定義がされなかったときに発生します)の場合、リストボックスは何も表示しません。

セレクションタイプのリストボックス

このタイプのリストボックスでは、列ごとにフィールドや式を割り当てます。それぞれの行はセレクションのレコードを基に評価されます。セレクションはカレントセレクションまたは命名セレクションです。
データ ソースがカレントセレクションである場合、データベースに対して行われた変更は自動でリストボックスに反映され、またリストボックスへの変更も自動で データベースに適用されます。つまりカレントセレクションは常に両方で同じです。セレクションタイプのリストボックスでは、LISTBOX INSERT ROW と LISTBOX DELETE ROW コマンドを使用できないことに留意してください。

リストボックスの列に式を割り当てることができます。式は1つ以上のフィールドから構成できます (例えば [Employees]LastName+" "+[Employees]FirstName)。または単にフォーミュラも使用できます (例えば String(Milliseconds))。式にはプロジェクトメソッド、変数、配列要素も指定できます。

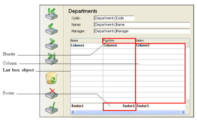
LISTBOX SET TABLE SOURCE コマンドを使用して、リストボックスに関連付けるテーブルを変更できます。

オブジェクト、列、ヘッダー、フッター

リストボックスオブジェクトは、以下4つの項目で構成されます:

- **オブジェクト自体**
- **列**
- **列ヘッダー** (表示/非表示制定可能、デフォルトで表示)
- **列フッター** (表示/非表示制定可能、デフォルトで表示)

これらの項目はフォームエディタ上では個別に選択できます。それぞれが独自のオブジェクト名や変数名を持ち、個別に処理されます。



リストボックスオブジェクトとは別に、各オブジェクトには以下のような名前が与えられます: 列はColumnN、ヘッダーはHeaderN、フッターはFooterN (Nは番号)。

各 項目タイプには独自の特性ならびに他の項目と共有する特性があります。例えば文字のフォントはリストボックスオブジェクトに一括して割り当てることも、列 やヘッダー、フッターに対して個別に割り当てることもできます。これとは逆に、入力プロパティは列に対してのみ指定することができます。

このルールはリストボックスに使用できる "オブジェクトプロパティ" テーマのコマンドに対して適用されます。その機能に応じて、各コマンドをリストボックスや列、ヘッダー、フッターに対して使用します。作業を行おうとする項目のタイプを設定するには、その項目に関連付けた名前や変数を渡します。

次の表は、リストボックスに使用できる "オブジェクトプロパティ" テーマのそれぞれのコマンドのスコープの詳細についてのまとめです:

オブジェクトプロパティコマンド	オブジェクト	列	列ヘッダー	列フッター
OBJECT MOVE	O			
OBJECT GET COORDINATES	O			
OBJECT SET RESIZING OPTIONS	O			
OBJECT GET RESIZING OPTIONS	O			
OBJECT SET FILTER		O		
OBJECT SET FORMAT		O		O
OBJECT SET ENTERABLE		O		
OBJECT SET CHOICE LIST NAME		O		
OBJECT SET TITLE			O	
OBJECT SET COLOR	O	O	O	O
OBJECT SET RGB COLORS	O	O	O	O
OBJECT SET FONT	O	O	O	O
OBJECT SET FONT SIZE	O	O	O	O
OBJECT SET FONT STYLE	O	O	O	O
OBJECT SET HORIZONTAL ALIGNMENT	O	O	O	O
OBJECT Get horizontal alignment	O	O	O	O
OBJECT SET VERTICAL ALIGNMENT	O	O	O	O
OBJECT Get vertical alignment	O	O	O	O
OBJECT SET VISIBLE	O	O	O	O
OBJECT SET SCROLLBAR	O			
OBJECT GET BEST SIZE	O	O	O	

注: 配列型のリストボックスではスタイル、フォントカラー、背景色、行ごとの表示を個別に指定できます。これはリストボックスのプロパティリストを使用して関連付けた配列を通して行います。これらの配列名はLISTBOX GET ARRAYS コマンドを使用して取得できます。

リストボックスとランゲージ

オブジェクトメソッド

リストボックスオブジェクトやリストボックスの各列に対し、オブジェクトメソッドを付加することができます。オブジェクトメソッドの呼び出しは、次の順で行われます:

1. 各列のオブジェクトメソッド
2. リストボックスのオブジェクトメソッド

ヘッダーとフッターで発生したイベントは、その列のオブジェクトメソッドが受け取ります。

OBJECT SET VISIBLEとヘッダー/フッター

ヘッダーやフッターにOBJECT SET VISIBLEコマンドを使用すると、このコマンドに渡した引数に関わらず、そのリストボックス中のすべてのヘッダーやフッターが対象になります。例えばOBJECT SET VISIBLE(*;"header3";False)という命令の場合、指定したヘッダーだけではなく、header3が属するリストボックスの全ヘッダーを非表示にします。
OBJECT SET VISIBLEコマンドを使用してこれらのオブジェクトの表示/非表示を管理できるようにするには、フォームエディターレベルで表示に設定されていなければなりません (**ヘッダーを表示**や**フッターを表示**オプションを選択する)。

OBJECT Get pointer

Object with focusやObject current定数とともに使用されるOBJECT Get pointer (以前のFocus objectとSelf) はリストボックスやリストボックス列のオブジェクトメソッドで使用できます。これらはフォームイベントのタイプに基づきリストボックス、リストボックス列(1)、ヘッダー変数、またはフッター変数へのポインターを返します。以下の表に動作をまとめます:

イベント	Object with focus	Object current
On Clicked	リストボックス	列
On Double Clicked	リストボックス	列
On Before Keystroke	列	列
On After Keystroke	列	列
On After Edit	列	列
On Getting Focus	列またはリストボックス (*)	列またはリストボックス (*)
On Losing Focus	列またはリストボックス (*)	列またはリストボックス (*)
On Drop	リストボックスソース	リストボックス (*)
On Drag Over	リストボックスソース	リストボックス (*)
On Begin Drag Over	リストボックス	リストボックス (*)
On Mouse Enter	リストボックス (**)	リストボックス (**)
On Mouse Move	リストボックス (**)	リストボックス (**)
On Mouse Leave	リストボックス (**)	リストボックス (**)
On Data Change	列	列
On Selection Change	リストボックス (**)	リストボックス (**)
On Before Data Entry	列	列
On Column Moved	リストボックス	列
On Row Moved	リストボックス	リストボックス
On Column Resize	リストボックス	列
On Open Detail	Nil	リストボックス (**)
On Close Detail	Nil	リストボックス (**)
On Header Click	リストボックス	ヘッダー
On Footer Click	リストボックス	フッター
On After Sort	リストボックス	ヘッダ

(*) リストボックス中でフォーカスが更新されると、列へのポインターが返されます。フォームレベル上でフォーカスが更新されると、リストボックスへのポインターが返されます。列のオブジェクトメソッドのコンテキストでは、列へのポインターが返されます。
(**) 列のオブジェクトメソッドのコンテキストでは実行されません。

(1) 列へのポインターが返された時、指し示すオブジェクトはリストボックスのタイプによります。配列型のリストボックスにおいて、OBJECT Get pointer("ユーザーインターフェース"テーマ)は、フォーカスを取得したリストボックスの列（つまり配列）へのポインターを返します。4Dのポインターのメカニズムを利用し、修正された配列の項目番号を調べることができます。例えば、ユーザーが列col2の5行目を変更した場合は、次の ようになります:

```
$Column:=OBJECT Get pointer
 // $Columnにはcol2へのポインタが含まれる
$Row:=$Column-> // $Row は 5
```

セレクション型のリストボックスで、OBJECT Get pointerは以下を返します:

- フィールドが関連付けられた列の場合、そのフィールドへのポインター
- 変数が関連付けられた列の場合、その変数へのポインター
- 式が関連付けられた列の場合、Nilポインター

OBJECT SET SCROLL POSITION

OBJECT SET SCROLL POSITION コマンド("オブジェクトプロパティ"テーマ)をリストボックスで使用できます。スクロールは、リストボックスの最初に選択された行または指定された行を表示させます。

EDIT ITEM

EDIT ITEMコマンド("入力制御"テーマ)を使用して、リストボックスオブジェクトのセルを編集モードに移行することができます。

REDRAW

REDRAWコマンド ("ユーザーインターフェーステーマ") がセレクション表示モードのリストボックスに適用されると、コマンドはリストボックス中に表示されたデータの更新を実行します。

Displayed line number

Displayed line number コマンド ("" テーマ) はリストボックスオブジェクトのOn Display Detail フォームイベントのコンテキストで動作します。

フォームイベント

リストボックス管理、特にドラッグ＆ドロップや並び替え操作を管理するために、特別なフォームイベントを使用できます。詳細については、Form Eventコマンドの節を参照してください。

ドラッグ＆ドロップ

リストボックス中でのデータのドラッグ＆ドロップ管理は、Drop position と DRAG AND DROP PROPERTIESコマンドでサポートされます。これらのコマンドは特にリストボックスに適用されます。

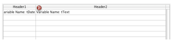
行や列のドラッグ＆ドロップと混同しないように注意してください。これらはLISTBOX MOVED ROW NUMBER や LISTBOX MOVED COLUMN NUMBER コマンドでサポートされます。

入力の管理

リストボックスセルを入力可能にするには、以下の条件を満たす必要があります:

- セルが属する列を**入力可**に設定する (そうしない場合、その列のセルには入力できません)。
- On Before Data Entry イベントで$0から-1を返さない。
  カーソルがセルに入ると、その列のメソッドでOn Before Data Entryイベントが生成されます。このイベントのコンテキストから$0に-1を設定すると、そのセルは入力不可として扱われます。イベントが**Tab** や **Shift+Tab**が押された後に生成されると、それぞれフォーカスは次あるいは前のセルに移動します。$0が-1でなければ (デフォルトは0)、列は入力化であり編集モードに移行します。

2つの配列で構築されるリストボックスを考えてみましょう。ひとつは日付でもう一つはテキストです。日付配列は入力不可に設定されていますが、テキスト配列は日付が過去でない場合に入力可とします。



arrText列のメソッドは以下の通りです:

```
Case of
   :(Form event=On Before Data Entry) // セルがフォーカスを得た
      LISTBOX GET CELL POSITION(*;"lb";$col;$row)
 // セルの特定
      If(arrDate{$row}<Current date) // 日付が昨日以前なら
         $0:=-1 // セルは入力不可
      Else
 // そうでなければ入力可
      End if
End case
```

注: 4D v13よりOn Before Data EntryイベントはOn Getting Focusより前に生成されます。

ソートの管理

ヘッダがクリックされると、デフォルトでリストボックスは自動的に標準的なカラムの並び替えを行います。標準的な並び替えとは、列の値を文字順に並べ替え、続けてクリックされると昇順/降順を交互に切り替えます。すべての列は常に自動で同期されます。

リストボックスの並び替え可プロパティの選択を解除すると、ユーザによる標準の並び替えを禁止することができます。

開発者は、LISTBOX SORT COLUMNSコマンドを使用するか、またはOn Header ClickとOn After Sortフォームイベント (Form eventコマンドの節を参照) と4Dの配列管理コマンドを組み合わせて、独自の並び替えを設定することができます。

Note: 列のプロパティ"並び替え可"は、ユーザによる標準の並び替えにのみ影響を与えます。LISTBOX SORT COLUMNSコマンドでは、このプロパティが考慮されません。

列ヘッダに関連付けられた変数の値を使用すると、列の現在の並び替え状況（読み込み）や並び替え矢印の表示など、追加情報を管理することができます。

- 変数が0のとき、列は並び替えられておらず、矢印は表示されていません;

- 変数が1のとき、列は昇順で並び替えられていて、並び替え矢印が表示されています;

- 変数が2のとき、列は降順で並び替えられていて、並び替え矢印が表示されています。

変数の値を設定して (例えばHeader2:=2)、ソートを表す矢印の表示を強制することができます。しかし列のソート順は変更されません、これを処理するのは開発者の役割です。

選択行の管理

選択行の管理は、リストボックスのタイプが配列かセレクションかにより異なります。

- **セレクションタイプのリストボックス**: 選択行はデフォルトでは、*$ListboxSetX*と呼ばれる修正可能なセットにより管理されます(*X*は0から始まり、フォーム内のリストボックスの数に応じて一つずつ増加していきます)。このセットはリストボックスのプロパティリストで定義します。このセットは4Dが自動 で管理します。ユーザがリストボックス中で1つ以上の行を選択すると、セットが即座に更新されます。他方、プログラムからリストボックスの選択を更新する た めに、"セット"テーマのコマンドを使用することができます。
- **配列タイプのリストボックス**: LISTBOX SELECT ROWコマンドを使用して、プログラムからリストボックスの行を選択できます。
  リストボックスオブジェクトにリンクされた変数は、オブジェクト行の選択の取得、設定、保存に使用します。この変数はブール配列で、4Dが自動的に作成・保守を行います。この配列のサイズは、リストボックスのサイズにより決定されます。配列には列に関連付けられた最も小さな配列と同じ数の要素が含まれます。
  この配列の各要素には、対応する行が選択された場合には**True**が、それ以外の場合は**False**が設定されます。4Dは、ユーザの動作に応じてこの配列の内容を更新します。これとは逆に、この配列要素の値を変更して、リストボックス中の選択行を変更することができます。
  この配列の各要素には、対応する行が選択された場合には**True**が、それ以外の場合は**False**が設定されます。4Dは、ユーザの動作に応じてこの配列の内容を更新します。これとは逆に、この配列要素の値を変更して、リストボックス中の選択行を変更することができます。
  他方、この配列への要素の挿入や削除はできず、行のタイプ変更もできません。

注: Count in arrayコマンドを使用して、選択された行の数を調べることができます。

例えば、以下のメソッドは配列タイプのリストボックスで、最初の行の選択を切り替えます:

```
  ` tBListBoxはフォーム上のリストボックス変数の名前
If(tBListBox{1}=True)
   tBListBox{1}:=False
Else
   tBListBox{1}:=True
End if
```

注: 階層モードのリストボックスの選択行管理については階層リストボックスの管理を参照してください。

リストボックスの印刷

4D v12より、リストボックスを印刷することができます。2つの印刷モード、フォームオブジェクトのようにリストボックスを印刷するために使用できるプレビューモードと、フォーム内でリストボックスオブジェクト自身の印刷を制御できる詳細モードがあります。フォームエディタでリストボックスオブジェクトに"印刷"アピアランスを適用できる点に留意してください。

プレビューモード

プレビューモードでのリストボックスの印刷は、標準の印刷コマンドや**プリント**メニューを使用して、リストボックスを含むフォームを直接印刷します。リストボックスはフォーム上に表示されている通りに印刷されます。このモードではオブジェクトの印刷をち密に制御することはできません。特に表示されている以上の行を印刷することはできません。

詳細モード

このモードでは、リストボックスの印刷はPrint objectコマンドを使用してプログラムにより実行されます。そのため、プロジェクトフォーム上のリストボックスのみを詳細モードで印刷できます。LISTBOX GET PRINT INFORMATION コマンドを使用してオブジェクトの印刷を制御できます。

このモードでは:

- 印刷する行数がオブジェクトの元の高さよりも少ない場合、リストボックスオブジェクトの高さは自動で減少させられます ("空白"行は印刷されません)。他方、オブジェクトの内容に基づき高さが自動で増大することはありません。実際に印刷されるオブジェクトのサイズはLISTBOX GET PRINT INFORMATIONコマンドで取得できます。
- リストボックスオブジェクトは"そのまま"印刷されます。言い換えれば、ヘッダーやグリッド線の表示、表示/非表示行など、現在の表示パラメタが考慮されます。
  これらのパラメタには印刷される最初の行も含みます。印刷を実行する前にSCROLL LINESを呼び出すと、リストボックスに印刷される最初の行はコマンドで指定した行になります。
- 自動メカニズムにより、表示可能な行以上の行数を含むリストボックスの印刷が容易になります。連続してPrint objectを呼び出して、それぞれの呼び出し毎に新しい行セットを印刷することができます。LISTBOX GET PRINT INFORMATION コマンドを使用して、印刷が行われている間の状態をチェックできます。

リストボックスにSQLクエリの結果を表示する

SQLクエリーの結果を直接配列タイプのリストボックスに表示することができます。これによりSQLクエリーの結果を素早く見る方法が提供されます。SELECTタイプのクエリーのみを使用できます。このメカニズムは外部SQLデータベースには使用できません。

この機能は以下の原則に基づいて動作します:

- クエリーの結果を受け取るリストボックスを作成します。リストボックスのデータソースは**配列**に設定しなければなりません。
- SELECTタイプのSQLクエリーを実行し、結果をリストボックスに割り当てた変数に受け取ります。Begin SQL/End SQL キーワードを使用できます (4Dランゲージリファレンス参照)。
- ユーザーはリストボックス列をソートしたり、更新したりできます。
- SELECTクエリーを実行するたびに、リストボックス列はリセットされます (複数のSELECTクエリーを実行して、リストボックスに行を追加することはできません)。
- SQLクエリーの結果受け取る列数と同じ数の列をリストボックスに用意することを推奨します。SELECTクエリーにより返される列数よりもリストボックスの列数が少ない場合、自動で列が追加されます。SELECTクエリーの結果よりも多い列数がある場合、不必要な列は自動で隠されます。
  注: 自動で追加された列には一時的な配列がバインドされます。この一時的な配列はフォームが閉じられるまで存在します。また一時的な変数が各ヘッダー用に作成されます。LISTBOX GET ARRAYSコマンドが呼び出されると、*arrColVars*引数には一時的な配列へのポインターが、*arrHeaderVars*引数には一時的なヘッダー変数へのポインターが返されます。例えば5つの列が追加されると、5番目の列の配列名は*sql_column5*でヘッダー名は*sql_header5*となります。
- インタープリターモードでは、SQLクエリーにより返されたデータに基づき、自動でリストボックスの既存の配列が型変更される場合があります。

例

PEOPLEテーブルのすべてのフィールドのデータを取得し、*vlistbox*という変数名のリストボックスに結果を表示します。これを行うためのメソッドは以下の通りです:

```
Begin SQL
   SELECT * FROM PEOPLE INTO <<vlistbox>>
End SQL
```

階層リストボックスの管理

4Dで階層リストボックスを使用できます。階層リストボックスとは、一列目の内容が階層形式で表示されるリストボックスのことです。このタイプの表現は繰り返される値や、(国、地域、都市など) 階層的に表現される値を含む情報の表示に使用されます。

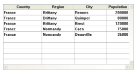
配列タイプのリストボックスのみが階層になることができます。

階層リストボックスはデータを表示する特別な方法ですが、データの構造 (配列) を変更することはありません。階層リストボックスは通常のリストボックスとまったく同様に管理されます (リストボックスオブジェクトの管理を参照)。

階層リストボックスを指定するには、2つの異なる方法があります:

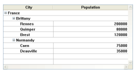
- フォームエディタのプロパティリストやリストボックス管理ポップアップメニューを使用して、手作業で階層要素を指定する。この方法は*Design Reference*で説明します。
- LISTBOX SET HIERARCHYとLISTBOX GET HIERARCHYコマンドを使用する。

階層リストボックスを含むフォームが最初に開かれるとき、デフォルトですべての行が展開されて表示されます。
配列中で値が繰り返されているとき、ブレーク行と階層ノードが自動でリストボックスに追加されます。例えば国、地域、名前、そして人口データを持つ市区が、リストボックスに4 つの配列で含まれているとします:



Country	Region	City	Population
France	Brittany	Rennes	200000
France	Brittany	Quimper	80000
France	Brittany	Brest	120000
France	Normandy	Caen	75000
France	Normandy	Deauville	35000

このリストボックスが階層形式で表示されると、最初の3つの配列は階層に含まれ、以下のように表示されます:



City	Population
⊟ France	
⊟ Brittany	
Rennes	200000
Quimper	80000
Brest	120000
⊟ Normandy	
Caen	75000
Deauville	35000

階層が構築される前に配列はソートされていません。例えばもし配列がデータAAABBAACCで構成されていると、取得される階層は以下のようになります:

> A
> B
> A
> C

階層ノードを展開あるいは折りたたむにはノードをクリックします。**Alt+click** (Windows) あるいは **Option+click** (Mac OS) を行うとすべてのサブ要素が展開されたり折りたたまれたりします。これらの操作はLISTBOX EXPAND と LISTBOX COLLAPSEコマンドを使用してプログラムで行うこともできます。

選択行とその位置の管理

階層リストボックスはノードの展開/ 折りたたみ状態により、スクリーン上に表示される行数が変わります。しかし配列の行数が変わるわけではありません。表示が変わるだけでデータに変更はありません。

この原則を理解することは重要です。階層リストボックスに対するプログラムによる管理は常に配列データに対して行われるのであり、表示されたデータに対して行われるわけではないからです。特に、自動で追加されるブレーク行は、表示オプション配列では考慮されません (後述ブレーク行の管理参照)。

例として以下の配列を見てみましょう:

France	Brittany	Brest
France	Brittany	Quimper
France	Brittany	Rennes

これらの配列が階層的に表示されると、2 つのブレーク行が追加されるため、"Quimper" 行は2 行目ではなく4 行目に表示されます:

France
Brittany
Brest
Quimper
Rennes

階層であってもなくても、リストボックスにどのようにデータが表示されているかにかかわらず、"Quimper" が含まれる行を太字にしたい場合はステートメントStyle{2} = bold を使用しなければなりません。配列中の行の位置のみが考慮されます。

この原則は以下のものを管理する内部的な配列に適用されます:

- カラー
- 背景色
- スタイル
- 非表示行
- 選択行

例えばRennesを含む行を選択したい場合、以下のように書きます:

```
->MyListbox{3}:=True
```

非階層表示:

France	Brittany	Brest
France	Brittany	Quimper
France	Brittany	Rennes

階層表示:

France
Brittany
Brest
Quimper
Rennes

注: 親が折りたたまれているために行が隠されていると、それらは選択されなくなります。(直接あるいはスクロールで) 表示されている行のみを選択できます。言い換えれば行を選択かつ隠された状態にすることはできません。

選択と同様、LISTBOX GET CELL POSITION コマンドは階層リストボックスと非階層リストボックス同じ値を返します。つまり以下の両例題で、LISTBOX GET CELL POSITION は同じ位置 (3;2) を返します。

非階層表示:

France	Brittany	Brest	120000
France	Brittany	Quimper	80000
France	Brittany	Rennes	200000
France	Normandy	Caen	75000

階層表示:

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000

ブレーク行の管理

ユーザがブレーク行を選択すると、LISTBOX GET CELL POSITIONは 対応する配列の最初のオカレンスを返します。以下のケースで:

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000

LISTBOX GET CELL POSITIONは (2;4) を返します。プログラムでブレーク行を選択するにはLISTBOX SELECT BREAKコマンドを使用します。

ブレーク行はリストボックスのグラフィカルな表示 (スタイルやカラー) を管理する内部的な配列では考慮されません。しかしオブジェクトのグラフィックを管理するオブジェクト(フォーム)テーマのコマンドを使用してブレーク行の表示を変更できます。階層を構成する配列に対して、適切なコマンドを実行します。

以下のリストボックスを例題とします (割り当てた配列名は括弧内に記載しています):

非階層表示:

(T1)	(T2)	(T3)	(T4)	(tStyle)	(tColor)
France	Brittany	Brest	120000	*Normal*	*0*
France	Brittany	Quimper	80000	*Underline*	*0*
France	Brittany	Rennes	200000	*Normal*	*0xFF0000*
France	Normandy	Caen	220000	*Normal*	*0*
France	Normandy	Deauville	4000	*Normal*	*0*

階層表示:

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000
Deauville	4000

階層モードでは*tStyle*や*tColors*配列で変更されたスタイルは、ブレーク行に適用されません。ブレークレベ ルでカラーやスタイルを変更するには、以下のステートメントを実行します:

```
OBJECT SET RGB COLORS(T1;0x0000FF;0xB0B0B0)
OBJECT SET FONT STYLE(T2;Bold)
```

注: このコンテキストでは、配列は割り当てられたオブジェクトがないため、配列変数を使用したシンタックスのみがオブジェクトプロパティコマンドで動作します。

結果:

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000
Deauville	4000

非表示行

サブ階層のすべての行が隠されているとき、ブレーク行は自動で隠されます。先の例題で1から3行目までが隠されていると、"Brittany"ブレーク行は表示されません。

展開/折りたたみ管理の最適化

On Expand や On Collapse フォームイベントを使用して階層リストボックスの表示を最適化できます。

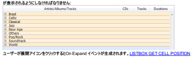
階層リストボックスはその配列の内容から構築されます。そのためこれらの配列すべてがメモリにロードされる必要があります。大量のデータから (SELECTION TO ARRAYを使用して) 生成される配列を元に階層リストボックスを構築するには、表示速度だけでなくメモリ利用量の観点からも困難が伴います。

On Expand と On Collapse フォームイベントを使用してこの制限を回避できます。例えばユーザーのアクションを元に階層の一部だけを表示したり、必要に応じて配列をロード/アンロードできます。

これらのイベントのコンテキストでは、LISTBOX GET CELL POSITIONコマンドは、行を展開/折りたたむためにユーザーがクリックしたセルを返します。

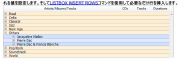
この場合、開発者がコードを使用して配列を空にしたり値を埋めたりしなければなりません。実装すべき原則は:

- リストボックスが表示される際、先頭の配列のみ値を埋めます。しかし2番目の配列を空の値で生成し、リストボックスに展開/折りたたみアイコンが表示されるようにしなければなりません:



Artists/Albums/Tracks	CDs	Tracks	Durations
⊞ Brasil			
⊞ Celtic			
⊞ Classical			
⊞ Jazz			
⊞ New Age			
⊞ Others			
⊞ Pop/Rock			
⊞ Soundtrack			
⊞ World			

- ユーザーが展開アイコンをクリックするとOn Expand イベントが生成されます。LISTBOX GET CELL POSITION コマンドはクリックされたセルを返すので、適切な階層を構築します: 先頭の配列に繰り返しの値を設定し、2番目の配列には SELECTION TO ARRAY コマンドから得られる値を設定します。そしてLISTBOX INSERT ROWSコマンドを使用して必要なだけ行を挿入します。



Artists/Albums/Tracks	CDs	Tracks	Durations
⊞ Brasil			
⊞ Celtic			
⊞ Classical			
⊞ Jazz			
⊞ New Age			
⊟ Others			
⊞ Jacqueline Maillan			
⊞ Pierre Dac			
⊞ Pierre Dac & Francis Blanche			
⊞ Pop/Rock			
⊞ Soundtrack			
⊞ World			

- ユーザーが折りたたみアイコンをクリックすると On Collapse イベントが生成されます。LISTBOX GET CELL POSITION コマンドはクリックされたセルを返すので、LISTBOX DELETE ROWS コマンドを使用してリストボックスから必要なだけ行を削除します。

LISTBOX COLLAPSE

LISTBOX COLLAPSE ({* ;} object {; recursive {; selector {; row | level {; column}}}})

引数	型		説明
*	演算子	→	指定時、objectはオブジェクト名 (文字列) 省略時、objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
recursive	ブール	→	True = サブレベルを折りたたむ False = サブレベルを折りたたまない
selector	倍長整数	→	折りたたむリストボックスのパーツ
row | level	倍長整数	→	折り畳むブレーク行の番号、または折り畳むリストボックスレベルの番号
column	倍長整数	→	折り畳むブレーク列の番号

説明

LISTBOX COLLAPSE コマンドを使用して objectと* で指定したリストボックスのブレーク行を折りたたみます。

オプションの* 引数を渡した場合、object引数はオブジェクト名 (文字列) です。この引数を渡さない場合objectは変数です。この場合、文字列ではなく変数参照を渡します。

リストボックスが階層モードとして設定されていない場合、コマンドはなにも行いません。階層リストボックスに関する詳細は[階層リストボックスの管理]を参照してください。

オプションのrecursive引数を使用してリストボックスの階層サブレベルの折りたたみを指定できます。Trueを渡すか省略すると、すべてのレベルおよびすべてのサブレベルが折りたたまれます。Falseを渡すと一番目のレベルのみが折りたたまれます。

オプションのselector引数を使用して、コマンドのスコープを指定できます。この引数にはテーマの以下の定数のいずれかを渡すことができます:

定数	型	値	コメント
Listbox all	倍長整数	0	コマンドはすべてのサブレベルに作用します (引数省略時のデフォルト値)。
Listbox selection	倍長整数	1	コマンドは選択されたサブレベルに作用します。
Listbox break row	倍長整数	2	コマンドはrowとcolumn引数で指定された"セル"に属するサブレベルに作用します。これらの引数は標準モードのリストボックスの行および列番号を表すことに留意してください。階層表現ではありません。rowと column 引数が省略されると、コマンドは何も行いません。
Listbox level	倍長整数	3	コマンドはlevel列に対応するすべてのブレーク行に作用します。この引数は標準モードのリストボックスの列番号を指定し、階層表現を考慮しません。level引数が省略されると、コマンドはなにも行いません。

選択あるいはリストボックスがブレーク行を含んでいないか、すべてのブレーク行がすでに折りたたまれている場合、コマンドはなにも行いません。

例題

この例はリストボックス中、選択されたブレーク行の第一レベルを折りたたみます:

```
LISTBOX COLLAPSE(*;"MyListbox";False;Listbox selection)
```

LISTBOX DELETE COLUMN

LISTBOX DELETE COLUMN ({* ;} object ; colPosition {; number})

引数	型		説明
*	演算子	⇒	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または 変数 (* 省略時)
colPosition	倍長整数	⇒	削除する列番号
number	倍長整数	⇒	削除する列数

説明

LISTBOX DELETE COLUMN コマンドは、引数objectおよび * で指定されたリストボックスから1つ以上の列 (表示または非表示) を取り除きます。

Note: このコマンドは階層モードで表示されているリストボックスの先頭の列に適用された場合なにも行いません。

オプション引数 * を渡すことにより、object引数がオブジェクト名 (文字列) であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はの節を参照してください。

オプション引数numberを渡さない場合、コマンドはcolPosition引数で指定された列だけを削除します。
この引数を渡した場合、number引数は、colPosition引数より、この列を含め右側にある削除する列の数を示します。

colPosition 引数がリストボックスの列数よりも大きい場合、コマンドは何も行いません。

LISTBOX DELETE ROWS

LISTBOX DELETE ROWS ({* ;} object ; vPosition {; numRows})

引数	型		説明
*	演算子	⇒	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または 変数 (* 省略時)
vPosition	倍長整数	⇒	削除する行の位置
numRows	倍長整数	⇒	削除する行の数

説明

LISTBOX DELETE ROWSコマンドは、object引数および * で指定されたリストボックスから、vPositionから始まるひとつ以上の行（表示または非表示）を削除します。

Note: このコマンドは配列タイプのリストボックスでのみ動作します。このコマンドがセレクションタイプのリストボックスに適用された場合、何も行わず、システム変数OKに0が設定されます。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

コマンド実行後、リストボックス内の要素の選択はすべて解除される点に留意してください。

vPositionで指定した行は、リストボックスの列で使用されるすべての配列から自動的に削除されます。
vPositionの値がリストボックス内の合計行数よりも大きい場合、コマンドは何も行いません。

Note: このコマンドはリストボックス行の表示/非表示状態を考慮しません。

LISTBOX DUPLICATE COLUMN

LISTBOX DUPLICATE COLUMN ({* ;} object ; positionCol ; colName ; colVariable ; headerName ; headerVar {; footerName ; footerVar})

引数	型		説明
*	演算子	⇨	指定時、Object はオブジェクト名 (文字列) 省略時、Object は変数
object	フォームオブジェクト	⇨	複製したい列のオブジェクト名(* 指定時)、または変数 (* 省略時)
positionCol	倍長整数	⇨	新しく複製した列の位置
colName	文字	⇨	新しい列の名前
colVariable	配列, フィールド, 変数	⇨	列の配列変数またはフィールド、変数
headerName	文字	⇨	列のヘッダーのオブジェクト名
headerVar	整数変数	⇨	列のヘッダーの変数
footerName	文字	⇨	列のフッターのオブジェクト名
footerVar	整数変数	⇨	列のフッターの変数

説明

LISTBOX DUPLICATE COLUMNコマンドは、objectと * 演算子によって指定された列を、プログラムによって実行中のフォームにおいて複製します。(アプリケーションモード)

注: 列を複製する機能は以前の4D にもありましたが、デザインモードにおいてフォームエディターのコンテキストメニューの中にある列複製コマンドを使用することによってのみ可能、というものでした。

複製元となる列においてプロパティリストやオブジェクト管理コマンド (OBJECT SET COLOR 等) を使用して設定されたスタイルオプション(サイズ、背景色、書式等)は、複製先の列にも反映されます。また、フォームのオブジェクトメソッド、スタイルとカラー配列やイベントなどといったものも複製されます。
しかしながら、データソース (リストボックスに定義されたソースにより、配列またはセレクション) に加えてスタイル配列、カラー配列は複製されません。列を複製した際には、デベロッパが新しい列にてそれらのデータを定義しなおさなければなりません。

objectと * 演算子を使用して複製する列を指定します。任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数で列変数を指定します。

注: このコマンドは階層リストボックスの最初の列を指定した場合には何もしません。

複製した列は、colPosition 引数で指定した位置の一つ前に置かれます。colPosition が列全体の総数より大きい場合、複製した列は最後の列の後ろに置かれます。

colName 引数と colVariable 引数には、オブジェクト名と、新しく複製された列の変数を渡します。

- 配列型のリストボックスの場合、変数名は列に表示される配列です。
- セレクション型のリストボックスの場合、colVariable 引数で指定したフィールドまたは変数の値が列に表示されます。これはリストボックスに関連付けられたセレクションのレコードごとに評価された値となります。この型の内容をしようするためには、リストボックスの「データソース」プロパティがカレントセレクションか命名セレクションに設定されている必要があります。

繰り返しになりますが、オリジナルの列のデータソースまでは複製されないことにご注意ください。複製された列に対し、新たにソースとなる変数、配列、フィールドを設定してあげる必要があります。

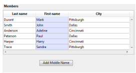
headerName と headerVariable 引数には、新たに作成する列のヘッダーのオブジェクト名と変数を渡します。同様に、footerName 引数と footerVariable 引数にオブジェクト名と変数を渡してあげることで新しい列のフッターを設定することもできます。footerVariable 引数省略時には 4D はフォーム変数を割り当てます。

Note: オブジェクト名は、フォーム内において固有でなければなりません。colName 引数、headerName 引数、footerName 引数などで渡す名前がこれまでに使用されていないことを確認して下さい。重複していた場合、複製は行われず、エラーが発生します。

このコマンドはフォーム表示中に使用される必要があります。通常、フォームの On Load イベント内か、ユーザーのアクションへの反応(On Clicked イベント) にて使用されます。

例題

以下のような配列型のリストボックスの中で、"First Name" の列を複製して入力できるようにしたい場合:

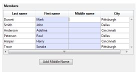

Members

Last name	First name	City
Durant	Mark	Pittsburgh
Smith	John	Dallas
Anderson	Adeline	Cincinnati
Peterson	Paul	Dallas
Harper	Harry	Cincinnati
Trace	Sandra	Pittsburgh

ボタンのコードは以下のようになります:

```
ARRAY TEXT(arrFirstNames2;Records in table([Members]))
LISTBOX DUPLICATE COLUMN(*;"column2";3;"col2bis";arrFirstNames2;"FirstNameA";vHead2A)
OBJECT SET TITLE(*;"FirstNameA";"Middle Name")
EDIT ITEM(*;"col2A";0)
```

ボタンをクリックすると、リストボックスは以下のようになります:


Members

Last name	First name	Middle name	City
Durant	Mark		Pittsburgh
Smith	John		Dallas
Anderson	Adeline		Cincinnati
Peterson	Paul		Dallas
Harper	Harry		Cincinnati
Trace	Sandra		Pittsburgh

LISTBOX EXPAND

LISTBOX EXPAND ({* ;} object {; recursive {; selector {; row | level {; column}}}})

引数	型		説明
*	演算子	→	指定時、objectはオブジェクト名 (文字列) 省略時、objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
recursive	ブール	→	True = サブレベルを展開 False = サブレベルを展開しない
selector	倍長整数	→	展開するリストボックスのパーツ
row | level	倍長整数	→	展開するブレーク行の番号、または 展開するリストボックスレベルの番号
column	倍長整数	→	展開するブレーク列の番号

説明

LISTBOX EXPAND コマンドはobjectと*で指定したリストボックスオブジェクトのブレーク行を展開するために使用します。

オプションの* 引数を渡した場合、object引数はオブジェクト名 (文字列) です。この引数を渡さない場合objectは変数です。この場合、文字列ではなく変数参照を渡します。

リストボックスが階層モードとして設定されていない場合、コマンドはなにも行いません。階層リストボックスに関する詳細は階層リストボックスの管理を参照してください。

オプションのrecursive引数を使用してリストボックスの階層サブレベルの展開を指定できます。Trueを渡すか省略すると、すべてのレベルおよびすべてのサブレベルが展開されます。Falseを渡すと指定された一番目のレベルのみが展開されます。

オプションのselector引数を使用して、コマンドのスコープを指定できます。この引数にはList Boxテーマの以下の定数のいずれかを渡すことができます:

定数	型	値	コメント
Listbox all	倍長整数	0	コマンドはすべてのサブレベルに作用します (引数省略時のデフォルト値)。
Listbox selection	倍長整数	1	コマンドは選択されたサブレベルに作用します。
Listbox break row	倍長整数	2	コマンドはrowと column引数で指定された"セル"に属するサブレベルに作用します。これらの引数は標準モードのリストボックスの行および列番号を表すことに留意してください。階層表現ではありません。rowと column 引数が省略されると、コマンドは何も行いません。
Listbox level	倍長整数	3	コマンドはlevel列に対応するすべてのブレーク行に作用します。この引数は標準モードのリストボックスの列番号を指定し、階層表現を考慮しません。level引数が省略されると、コマンドはなにも行いません。

このコマンドはブレーク行を選択しません。

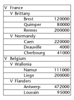
選択あるいはリストボックスがブレーク行を含んでいないか、すべてのブレーク行がすでに展開されている場合、コマンドはなにも行いません。

例題

以下の例題では、コマンドのさまざまな利用方法を示します。以下の配列がリストボックスに表示されているものとします:

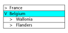
France	Brittany	Brest	120000
France	Brittany	Quimper	80000
France	Brittany	Rennes	200000
France	Normandy	Caen	220000
France	Normandy	Deauville	4000
France	Normandy	Cherbourg	41000
Belgium	Wallonia	Namur	111000
Belgium	Wallonia	Liege	200000
Belgium	Flanders	Antwerp	472000
Belgium	Flanders	Louvain	95000

```
  //リストボックスのすべてのブレーク行とサブ行を展開する
 LISTBOX EXPAND(*;"MyListbox")
```



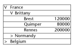
V France	
V Brittany	
Brest	120000
Quimper	80000
Rennes	200000
V Normandy	
Caen	220000
Deauville	4000
Cherbourg	41000
V Belgium	
V Wallonia	
Namur	111000
Liege	200000
V Flanders	
Antwerp	472000
Louvain	95000

```
  //選択された第一レベルのブレーク行を展開する
 LISTBOX EXPAND(*;"MyListbox";False;Listbox selection)
  //"Belgium"行が選択されている場合
```



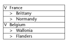
> France
V Belgium
> Wallonia
> Flanders

```
  //Brittanyブレーク行を展開し、サブレベルは展開しない
 LISTBOX EXPAND(*;"MyListbox";False;Listbox break row;1;2)
```



V France	
V Brittany	
Brest	120000
Quimper	80000
Rennes	200000
> Normandy	
> Belgium	

```
  //一番目の列 (国) のみを展開
 LISTBOX EXPAND(*;"MyListbox";False;Listbox level;1)
```



V France
> Brittany
> Normandy
V Belgium
> Wallonia
> Flanders

LISTBOX Get array

LISTBOX Get array ({* ;} object ; arrType) -> 戻り値

引数	型		説明
*	演算子	→	指定時:objectはオブジェクト名(文字列)省略時:objectは変数
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数(* 省略時)
arrType	倍長整数	→	配列のタイプ
戻り値	ポインター	↩	プロパティに関連付けられた配列へのポインター

説明

注: このコマンドは配列型のリストボックスに対してのみ有効です。

LISTBOX Get arrayコマンドは、objectと * によって指定されたリストボックスまたはリストボックスコラムの、スタイルまたは色に関連付けられた配列へのポインターを返します。

スタイル、カラー、背景色の配列は、デザインモードのプロパティリストを使用するか、新しいコマンド LISTBOX SET ARRAY 使用することで配列型のリストボックスと関連付けすることができます。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数で変数を指定します。対象がリストボックスなのか列なのかを、object 引数で指定します。

arrType 引数には、取得したいプロパティの配列の型を渡します。"List Box" テーマ内にある以下の定数を使用することができます。

定数	型	値
Listbox background color array	倍長整数	1
Listbox font color array	倍長整数	0
Listbox style array	倍長整数	2

戻り値は以下のどれかになります:

- 要求されたプロパティの配列がそのリストボックスまたは列に関連付けされていない場合、Nilが返ってきます。
- ユーザーによって定義されたプロパティの配列へのポインター
- LISTBOX SET ROW COLOR や LISTBOX SET ROW FONT STYLE コマンドを使用して設定されたプロパティの配列へのポインター

例題

典型的な使用例:

```
vPtr:=LISTBOX Get array(*;"MyLB";Listbox font color array)
 // "MyLB" というリストボックスに関連付けされた
 // フォントカラー配列を返します。

vPtr:=LISTBOX Get array(*;"Col4";Listbox style array)
 // "Col4" リストボックスの列に関連付けされた
 // フォントスタイル配列を返します。
```

LISTBOX GET ARRAYS

LISTBOX GET ARRAYS ({* ;} object ; arrColNames ; arrHeaderNames ; arrColVars ; arrHeaderVars ; arrColsVisible ; arrStyles {; arrFooterNames ; arrFooterVars})

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
arrColNames	文字配列	←	列オブジェクト名
arrHeaderNames	文字配列	←	ヘッダーオブジェクト名
arrColVars	ポインター配列	←	列変数へのポインターまたは列フィールドへのポインターまたはNil
arrHeaderVars	ポインター配列	←	ヘッダー変数へのポインター
arrColsVisible	ブール配列	←	列毎の表示状態
arrStyles	ポインター配列	←	配列へのポインターまたは スタイル、カラー、および表示変数またはNil
arrFooterNames	文字配列	←	列フッターオブジェクト名
arrFooterVars	ポインター配列	←	列フッター変数へのポインター

説明

LISTBOX GET ARRAYSコマンドは一連の同期化された配列を返し、object引数および * で指定されたリストボックスの各列（表示または非表示）に関する情報を提供します。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

コマンドを実行すると:

- arrColNames配列には、リストボックス内の各列のオブジェクト名一覧が代入されます。
- arrHeaderNames配列には、リストボックス内の各列ヘッダのオブジェクト名一覧が代入されます。
- arrColVars配列には、リストボックス内の各列に関連付けられた変数（配列）へのポインタが代入されます。セレクションタイプのリストボックスの場合、arrColVars には:
  - フィールドが関連付けられた列の場合、フィールドへのポインタ
  - 変数が関連付けられた列の場合、変数へのポインタ
  - 式が関連付けられた列の場合、Nilポインタが格納されます。
- arrHeaderVars配列には、リストボックス内の各列ヘッダに関連付けられた変数へのポインタが代入されます。
- arrColsVisible配列には各列に関するブール値が代入され、リストボックス内の列が表示（True）または非表示（False）のいずれであるかを示します。
- arrStyles配列には、4つの配列をそれぞれ指す4つのポインタが代入されます。これら4つの配列は、リストボックス内の各行に対してスタイルやフォントカラー、背景色、各行の非表示ステートを適用するために使用されます。これらの配列はデザインモードで、リストボックスのプロパティリストで指定されたものです。リストボックスに対する配列が指定されていない場合、arrStyles内の対応する項目にはNilポインタが代入されます。セレクションタイプのリストボックスでは、arrStyles には:
  - 変数により設定された設定ごとに、変数へのポインタが、
  - 式により設定された設定ごとに、Nilポインタが格納されます。

LISTBOX GET CELL POSITION

LISTBOX GET CELL POSITION ({* ;} object ; column ; row {; colVar})

引数	型		説明
*	演算子	⇒	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または 変数 (* 省略時)
column	倍長整数	⇐	列番号
row	倍長整数	⇐	行番号
colVar	ポインター	⇐	列変数へのポインタ

説明

LISTBOX GET CELL POSITIONコマンドは、objectおよび * で指定されたリストボックスの最後にクリックされたセル、あるいはキーボードアクションで選択されたセルの位置をcolumnとrowに返します。
このコマンドは、リストボックスが入力不可に設定されている場合でも、クリックや選択アクションの位置情報を返します。

Note: row 引数に返される数値は、リストボックス行の表示/非表示状態を考慮に入れません。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。

オプション引数colVarには、列に割り当てられている変数（例えば配列）に対するポインタが返されます。

このコマンドは、次のいずれかのフォームイベントが発生するリストボックスに対してのみ、使用することができます:

- On Clicked と On Double Clicked
- On Before Keystroke と On After Keystroke
- On After Edit
- On Getting Focus と On Losing Focus
- On Data Change
- On Selection Change
- On Before Data Entry

上記以外のイベントで使用された場合、LISTBOX GET CELL POSITIONコマンドはcolumnとrowに0を返します。

このコマンドは、マウスクリック、キーボード、(On Getting Focusを生成する) EDIT ITEM コマンドによる選択または選択解除を考慮します。

リストボックスの選択行がキーボードの矢印キーで変更された場合、column引数には0が返されます。この場合、colVarが渡されていればNilが返されます。

このコマンドで返される値は、リストボックスの列ヘッダにおける右クリック（あるいはMac OS のcontrol+クリック）では更新されません。

LISTBOX Get column formula

LISTBOX Get column formula ({* ;} object) -> 戻り値

引数	型	説明
*	演算子	→ 指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	→ オブジェクト名 (* 指定時) または変数 (* 省略時)
戻り値	文字	↩ 列に割り当てられたフォーミュラ

説明

LISTBOX Get column formulaコマンドはobjectと* 引数で指定したリストボックス列に割り当てられたフォーミュラを返します。フォーミュラはリストボックスプロパティのデータソースが**カレントセレクション**または**命名セレクション**の場合のみ使用できます。列にフォーミュラが割り当てられていない場合、コマンドは空の文字列を返します。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。この引数にはリストボックス列を指定しなければなりません。

LISTBOX Get column width

LISTBOX Get column width ({* ;} object {; minWidth {; maxWidth}}) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
minWidth	倍長整数	←	列の最小幅 (ピクセル単位)
maxWidth	倍長整数	←	列の最大幅 (ピクセル単位)
戻り値	倍長整数	↩	列幅 (ピクセル単位)

説明

LISTBOX Get column width コマンドは、object引数および * で指定された列の幅（ピクセル単位）を返します。object引数には、リストボックスの列や列ヘッダを渡すことができます。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はの節を参照してください。

LISTBOX Get column width は列サイズ変更の制限値を minWidth と maxWidth に返すことができます。これらの制限は LISTBOX SET COLUMN WIDTH コマンドで設定できます。
列の最小や最大サイズが設定されていない場合、対応する引数には0が返されます。

LISTBOX Get footer calculation

LISTBOX Get footer calculation ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	⇒	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	グラフ変数	⇒	オブジェクト名 (* 指定時) または変数 (* 省略時)
戻り値	倍長整数	↩	計算タイプ

説明

LISTBOX Get footer calculationコマンドはobjectと* 引数で指定したリストボックスのフッターエリアに割り当てられた自動計算のタイプを返します。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。

object 引数を使用して以下を指定します:

- フッターエリアの変数またはオブジェクト名。この場合コマンドはこのエリアに割り当てられた自動計算のタイプを返します。
- リストボックス列の変数またはオブジェクト名。この場合コマンドはこの列のフッターエリアに割り当てられた自動計算のタイプを返します。

返される値をListbox Footer Calculationテーマの定数と比較することができます (LISTBOX SET FOOTER CALCULATIONコマンド参照)。

LISTBOX Get footers height

LISTBOX Get footers height ({* ;} object {; unit}) -> 戻り値

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
unit	倍長整数	→	高さを指定する単位: 0または省略時 = ピクセル、1 = 行
戻り値	倍長整数	↩	行の高さ

説明

LISTBOX Get footers heightコマンドはobjectと* 引数で指定したリストボックスのフッター行の高さを返します。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。リストボックスあるいはリストボックスのフッターいずれかを指定できます。

unit 引数を省略した場合、デフォルトで高さはピクセル単位で返されます。異なる単位を指定するには unit 引数に List Box テーマの以下の定数を渡します:

定数	型	値	コメント
Listbox lines	倍長整数	1	高さを行数で指定。4Dはフォント設定に応じて高さを計算します。
Listbox pixels	倍長整数	0	高さをピクセルで指定 (デフォルト)。

注: 行の高さの計算に関する詳細はデザインリファレンスを参照してください。

LISTBOX GET GRID

LISTBOX GET GRID ({* ;} object ; horizontal ; vertical)

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
horizontal	ブール	←	True: 表示、False: 非表示
vertical	ブール	←	True: 表示、False: 非表示

説明

LISTBOX GET GRIDコマンドはobjectと * 引数で指定したリストボックスオブジェクトの縦横グリッド線の表示/非表示状態をそれぞれ返します。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。

コマンドは horizontal と vertical に、対応する線が表示されているか (True)、いないか (False) を返します。デフォルトでグリッドは表示されています。

LISTBOX GET GRID COLORS

LISTBOX GET GRID COLORS ({* ;} object ; hColor ; vColor)

引数	型		説明
*	演算子	⇒	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または変数 (* 省略時)
hColor	倍長整数	⇐	横グリッドのRGBカラー値
vColor	倍長整数	⇐	縦グリッドのRGBカラー値

説明

LISTBOX GET GRID COLORSコマンドはobjectと* 引数で指定したリストボックスの縦横グリッドカラーをそれぞれ返します。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。

コマンドは hColor と vColor にRGBカラーの値を返します。RGBカラーに関する詳細はOBJECT SET RGB COLORSコマンドの説明を参照してください。

LISTBOX Get headers height

LISTBOX Get headers height ({* ;} object {; unit}) -> 戻り値

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
unit	倍長整数	→	高さを指定する単位: 0または省略時 = ピクセル、1 = 行
戻り値	倍長整数	↩	行の高さ

説明

LISTBOX Get headers heightコマンドはobjectと* 引数で指定したリストボックスのヘッダー行の高さを返します。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。リストボックスあるいはリストボックスのヘッダーいずれかを指定できます。

unit 引数を省略した場合、デフォルトで高さはピクセル単位で返されます。異なる単位を指定するには unit 引数に List Box テーマの以下の定数を渡します:

定数	型	値	コメント
Listbox lines	倍長整数	1	高さを行数で指定。4Dはフォント設定に応じて高さを計算します。
Listbox pixels	倍長整数	0	高さをピクセルで指定 (デフォルト)。

注: 行の高さの計算に関する詳細はデザインリファレンスを参照してください。

LISTBOX GET HIERARCHY

LISTBOX GET HIERARCHY ({* ;} object ; hierarchical {; hierarchy})

引数	型	説明
*	演算子	→ 指定時、objectはオブジェクト名 (文字列) 省略時、objectは変数
object	フォームオブジェクト	→ オブジェクト名 (* 指定時) または 変数 (* 省略時)
hierarchical	ブール	← True = 階層リストボックス False = 非階層リストボックス
hierarchy	ポインター配列	← ポインタの配列

説明

LISTBOX GET HIERARCHY コマンドを使用して objectと*で指定したリストボックスのプロパティが階層であるかどうかを知ることができます。

オプションの * 引数を渡した場合、object引数はオブジェクト名 (文字列) です。この引数を渡さない場合objectは変数です。この場合、文字列ではなく変数参照を渡します。

ブール型のhierarchical引数はリストボックスのモードが階層モードであるかないかを示します:

- 引数がTrueを返すと、リストボックスは階層モードです。
- 引数がFalseを返すと、リストボックスは (非階層の) 標準配列モードで表示されています。

リストボックスが階層モードであるとき、hierarchy引数には階層を設定するために使用されるリストボックスの配列へのポインタが返されます。

Note: 階層リストボックスが階層モードでないとき、コマンドはhierarchy配列の最初の要素に、リストボックスの最初の列の配列へのポインタを返します。

LISTBOX Get information

LISTBOX Get information ({* ;} object ; info) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
info	倍長整数	→	取得する情報
戻り値	倍長整数	↩	現在値

説明

LISTBOX Get informationコマンドは、引数objectおよび * で指定されたリストボックスオブジェクトの、現在の表示状態やヘッダサイズ、スクロールバーに関する各種情報を返します。

オプションの引数 * を渡すことにより、object引数がオブジェクト名 (文字列) であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

引数infoには、取得しようとする情報のタイプを示す値を渡します。この引数には値またはList Boxテーマの次の定数のいずれかを使用することができます:

定数	型	値	コメント
Listbox display footer	倍長整数	8	0 = 非表示 1 = 表示
Listbox display header	倍長整数	0	0=非表示, 1=表示
Listbox display hor scrollbar	倍長整数	2	0=非表示, 1=表示
Listbox display ver scrollbar	倍長整数	4	0=非表示, 1=表示
Listbox footer height	倍長整数	9	高さ (ピクセル)
Listbox header height	倍長整数	1	高さ (ピクセル)
Listbox hor scrollbar height	倍長整数	3	高さ (ピクセル)
Listbox hor scrollbar position	倍長整数	6	カーソルの位置 (ピクセル)
Listbox ver scrollbar position	倍長整数	7	カーソルの位置 (ピクセル)
Listbox ver scrollbar width	倍長整数	5	幅 (ピクセル)

- 最初の6つの定数を使用すると、リストボックスのサイズ計算に役立ちます。
- 定数Position listbox hor scrollbarとPosition listbox ver scrollbarを使用すると、**LISTBOX Get information**コマンドは、例えばウィンドウの隠された部分のサイズなど、元の位置からの相対で、スクロールカーソルの位置をピクセル単位で返します。デフォルトの位置では、0となります。例えば、行の高さの情報と組み合わせることで、リストボックスに表示されたコンテンツを見つけることができます。
- フッターが表示されている場合、ステートメント **LISTBOX Get information**(vLB;Listbox footer height) はLISTBOX Get footers heightコマンドと同じ値を返します。しかしフッターが表示されていなければ**LISTBOX Get information**は0を返し、LISTBOX Get footers heightは、この場合仮想的な、フッターの高さを返します。

例題

行の高さが20ピクセルのリストボックスがある時、以下のコードを実行します:

```
$scroll:=LISTBOX Get information(*;"Listbox";Position listbox ver scrollbar)
```

$scrollの値が200の場合、リストボックスの最初から11番目の行が表示されていることがわかります。(200/20=10、10行がスクロールにより隠されている)

LISTBOX Get locked columns

LISTBOX Get locked columns ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列)、省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
戻り値	倍長整数	↩	横スクロールしない列数

説明

LISTBOX Get locked columnsコマンドはobjectと * 引数で指定したリストボックスで横スクロールしない列数を返します。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。

列はプロパティリストやLISTBOX SET LOCKED COLUMNSコマンドを使用してロックできます。詳細はデザインリファレンスマニュアルを参照してください。

ロックされたエリア内でプログラムを使用して列が挿入されたり削除されたりすると、このコマンドから返される列数も変更されます。例えばロックされた列をひとつ削除すると、ロックされた列の数はひとつ減ります。同様にロックされたエリアにプログラムで列を挿入すると、この列は自動でロックされ、ロックされた列数は一つ増えます。

しかしコマンドは列の表示/非表示状態は考慮しません。

LISTBOX Get number of columns

LISTBOX Get number of columns ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
戻り値	倍長整数	↩	列数

説明

LISTBOX Get number of columns コマンドは、object引数および * で指定されたリストボックスに存在する列（表示または非表示）の合計数を返します。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はの節を参照してください。

LISTBOX Get number of rows

LISTBOX Get number of rows ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
戻り値	倍長整数	↩	行数

説明

LISTBOX Get number of rowsコマンドは、object引数および * で指定されたリストボックスの行の数を返します。

Note: LISTBOX Get number of rowsは行の表示/非表示状態を考慮に入れません。例えば、10行のうち先頭の9行か非表示のリストボックスで、LISTBOX Get number of rowsは10を返します。

オプションの引数*を渡すことにより、object引数がオブジェクト名 (文字列) であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はの節を参照してください。

Note: リストボックスのカラムに関連付けられたすべての配列が同じサイズではない場合、最小の配列サイズだけがリストボックスに表示され、またこのコマンドもその数値を返します。

LISTBOX GET OBJECTS

LISTBOX GET OBJECTS ({* ;} object ; arrObjectNames)

引数	型		説明
*	演算子	→	指定時、Object はオブジェクト名(文字列) 省略時、Object は変数
object	フォームオブジェクト	→	オブジェクト名(* 指定時)、または変数(* 省略時)
arrObjectNames	テキスト配列	←	リストボックスを構成するサブオブジェクト名(ヘッダー、列、フッター)

説明

LISTBOX GET OBJECTS コマンドは 、objectと* 演算子で指定したリストボックスを構成するオブジェクトの、それぞれの名前を含んだ配列を返します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数で変数を指定します。この場合、文字列ではなく変数参照を渡します。

arrObjectNames には、コマンドからの配列を受け取るテキスト配列を渡します。配列には、オブジェクト名が以下の規則に従って表示順に返されます:

nameCol1
headerNameCol1
footerNameCol1
nameCol2
headerNameCol2
footerNameCol2
...

配列には、表示非表示に関わらず、全ての列に関して(列のフッターを含む)オブジェクト名が返されます。

このコマンドは FORM LOAD、FORM GET OBJECTS と OBJECT Get type コマンドを使用してフォームを解析する際に有用です。必要に応じてリストボックスのサブオブジェクトの名前を取得するために使用することができます。

例題

フォームをロードし、そこに内包される全てのリストボックスオブジェクトの一覧を取得する場合:

```
FORM LOAD("MyForm")
ARRAY TEXT(arrObjects;0)
FORM GET OBJECTS(arrObjects)
ARRAY LONGINT(ar_type;Size of array(arrObjects))
ARRAY TEXT(ar_typeName;Size of array(arrObjects))
For($i;1;Size of array(arrObjects))
   ar_type{$i}:=OBJECT Get type(*;arrObjects{$i})
   If(ar_type{$i}=Object type listbox)
      ARRAY TEXT(arrLBObjects;0)
      LISTBOX GET OBJECTS(*;arrObjects{$i};arrLBObjects)
   End if
End for
FORM UNLOAD
```

LISTBOX GET PRINT INFORMATION

LISTBOX GET PRINT INFORMATION ({* ;} object ; selector ; info)

引数	型		説明
*	演算子	→	指定時、objectはオブジェクト名 (文字列) 省略時、objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
selector	倍長整数	→	取得する情報
info	倍長整数, ブール	←	現在の値

説明

LISTBOX GET PRINT INFORMATION コマンドは objectと*で指定したリストボックスオブジェクトの印刷に関連する現在の情報を返します。このコマンドを使用してリストボックスの内容の印刷を制御します。

オプションの * 引数を渡した場合、object引数はオブジェクト名 (文字列) です。この引数を渡さない場合objectは変数です。この場合、文字列ではなく変数参照を渡します。

このコマンドはPrint objectコマンドによるリストボックスの印刷のコンテキストで呼ばれなければなりません。このコンテキスト外では、意味のある値を返しません。

selectorに知りたい情報を示す値を渡し、infoには数値またはブール変数を渡します。selectorには"List Box"テーマの以下の定数を渡します:

定数	型	値	コメント
Listbox last printed row number	倍長整数	0	infoに印刷された最後の行番号を返します。これにより次に印刷される行の番号が分かります。 リストボックスに非表示行が存在したりOBJECT SET SCROLL POSITIONコマンドが呼び出されていたりすると、返される値は実際に印刷された行数よりも、大きくなることがあります。例えば行番号1, 18そして20が印刷されると、infoには20が返されます。
Listbox printed height	倍長整数	3	infoに実際に印刷されたオブジェクトの高さをピクセル単位で返します (ヘッダーや線等を含む)。印刷する行数がリストボックスの高さに満たない場合、高さは自動で減らされます。
Listbox printed rows	倍長整数	1	さいごのPrint object最後のコマンド呼び出し時に実際に印刷された行数をinfoに返します。この数値には階層リストボックスの場合に追加されたブレーク行も含まれます。例えばリストボックスに20行あり、奇数行が隠されている場合、infoは10になります。
Listbox printing is over	倍長整数	2	リストボックスの最後の (表示) 行が印刷されたかどうかを示すブール値をinfoに返します。True = 行は印刷された; そうでなければFalse。

リストボックスの印刷に関する原則についてはリストボックスの印刷を参照してください。

例題 1

すべての行が印刷されるまで印刷を行う:

```
OPEN PRINTING JOB
OPEN PRINTING FORM("SalesForm")

$Over:=False
Repeat
   $Total:=Print object(*;"mylistbox")
   LISTBOX GET PRINT INFORMATION(*;"mylistbox";Listbox printing is over;$Over)
Until($Over)

CLOSE PRINTING JOB
```

例題 2

特定の行が隠されていて、リストボックスを最低500行印刷する:

```
$GlobalPrinted:=0
Repeat
   $Total:=Print object(*;"mylistbox")
   LISTBOX GET PRINT INFORMATION(*;"mylistbox";Listbox printed rows;$Printed)
   $GlobalPrinted:=$GlobalPrinted+$Printed
   PAGE BREAK
Until($GlobalPrinted>=500)
```

LISTBOX Get row color

LISTBOX Get row color ({* ;} object ; row {; colorType}) -> 戻り値

引数	型	説明
*	演算子	➡指定時:objectはオブジェクト名(文字列)省略時:objectは変数
object	フォームオブジェクト	➡オブジェクト名(*指定時)、または変数(*省略時)
row	倍長整数	➡列番号
colorType	倍長整数	➡Listbox font color (デフォルト値) または Listbox background color
戻り値	倍長整数	⮐カラーの値

説明

注: このコマンドは配列型のリストボックスに対してのみ有効です。

LISTBOX Get row colorコマンドは、objectと * によって指定されたリストボックス内の、行またはセルの色を返します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数で変数を指定します。
対象がリストボックスなのか列なのかを、object 引数で指定します。

- object がリストボックス全体を指定している場合、コマンドは指定された行の色を返します。
- object がリストボックス内の列を指定している場合、コマンドは単一のセルの色を返します。

row 引数にはカラーを取得したい行を番号で指定します。

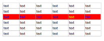
注: このコマンドは行の表示/非表示の状態は無視します。

colorType 引数には、行の背景色を取得したいのかフォントカラーを取得したいのかによって、Listbox background color または Listbox font color の定数のどちらかを渡します。省略時は、フォントカラーが返されます。

Warning: 行に割り当てられたカラーであっても、それが全てのセルに配色されるとは限りません(以下の例を参照のこと)。リストボックスまたはリストボックス列に対して矛盾する値がプロパティを使用して設定された場合、4D内部の優先順位が適用されます。詳細な情報に関しては *Design Reference* マニュアルを参照して下さい。

例題

以下に与えられたリストボックスについて考えます。



text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

```
vColor:=LISTBOX Get row color(*;"Col5";3)
vColor2:=LISTBOX Get row color(*;"List Box";3)
vColor3:=LISTBOX Get row color(*;"List Box";Listbox background color)
 // vColor contains 0xFFFF00 (yellow)
 // vColor2 contains 0x00FF (blue)
 // vColor3 contains 0x00FF0000 (red)
```

LISTBOX Get row font style

LISTBOX Get row font style ({* ;} object ; row) -> 戻り値

引数	型		説明
*	演算子	→	指定時:objectはオブジェクト名(文字列)省略時:objectは変数
object	フォームオブジェクト	→	オブジェクト名(*指定時)、または変数(*省略時)
row	倍長整数	→	列番号
戻り値	倍長整数	↩	スタイルの値

説明

注: このコマンドは配列型のリストボックスに対してのみ有効です。

LISTBOX Get row font styleコマンドは、objectと * によって指定されたリストボックス内の、行またはセルのフォントスタイルを返します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数で変数を指定します。
対象がリストボックスなのか列なのかを、object 引数で指定します。

- object がリストボックス全体を指定している場合、コマンドは指定された行のスタイルを返します。
- object がリストボックス内の列を指定している場合、コマンドは単一のセルのスタイルを返します。

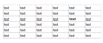
row 引数にはスタイルを取得したい行を番号で指定します。

注: このコマンドは行の表示/非表示の状態は無視します。

Warning: 行に割り当てられたスタイルであっても、それが全てのセルに適用されるとは限りません(以下の例を参照のこと)。リストボックスまたはリストボックス列に対して矛盾する値がプロパティを使用して設定された場合、4D内部の優先順位が適用されます。詳細な情報に関しては *Design Reference* マニュアルを参照して下さい。

例題

以下に与えられたリストボックスについて考えます。



text	text	text	text	text	text
text	text	text	text	text	text
text	*text*	*text*	*text*	**text**	*text*
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

```
vStyle:=LISTBOX Get row font style(*;"Col5";3)
vStyle2:=LISTBOX Get row font style(*;"List Box";3)
 // vStyle contains 1 (Bold)
 // vStyle2 contains 6 (Italic + Underline)
```

LISTBOX Get rows height

LISTBOX Get rows height ({* ;} object {; unit}) -> 戻り値

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
unit	倍長整数	→	高さを表す単位: 0または省略時はピクセル、1の場合行単位
戻り値	整数	↩	行の高さ

説明

LISTBOX Get rows heightコマンドは、object引数および * で指定されたリストボックスの現在の行の高さを返します。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

unit 引数を省略するとデフォルトで、行の高さはピクセル単位で行われます。他の単位を使用する場合は unit 引数にList Boxテーマの以下の定数を渡します:

定数	型	値	コメント
Listbox lines	倍長整数	1	高さを行数で指定。4Dはフォント設定に応じて高さを計算します。
Listbox pixels	倍長整数	0	高さをピクセルで指定 (デフォルト)。

注: 行の高さの計算に関する情報はデザインリファレンスマニュアルを参照してください。

LISTBOX Get static columns

LISTBOX Get static columns ({* ;} object) -> 戻り値

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
戻り値	倍長整数	↩	ドラッグで移動しない列数

説明

LISTBOX Get static columnsコマンドはobject と* 引数で指定したリストボックス中のドラッグで移動しない列数を返します。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。

スタティック列はプロパティリストやLISTBOX SET STATIC COLUMNSコマンドを使用して設定できます。

スタティック列内でプログラムを使用して列が挿入されたり削除されたりすると、このコマンドから返される列数も変更されます。
しかしコマンドは列の表示/非表示状態は考慮に入れません。

注: スタティック列とロックされた列は異なる機能です。詳細はデザインリファレンスマニュアルを参照してください。

LISTBOX GET TABLE SOURCE

LISTBOX GET TABLE SOURCE ({* ;} object ; tableNum {; name {; highlightName}})

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
tableNum	倍長整数	←	セレクションのテーブル番号
name	文字	←	命名セレクション名 またはカレントセレクションの場合""
highlightName	文字	←	ハイライトセット名

説明

LISTBOX GET TABLE SOURCEコマンドを使用して、objectと * 引数で指定したリストボックスに表示されるデータの現在のソースを知ることができます。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

このコマンドはリストボックスに関連付けられたメインテーブルのテーブル番号をtableNum引数に、使用される命名セレクション名をオプションのname 引数に返します。
リストボックス行がテーブルのカレントセレクションに関連付けられている場合、name 引数には空の文字列が返されます。リストボックス行がテーブルの命名セレクションに関連付けられている場合、name 引数にはこの命名セレクション名が返されます。

リストボックスが配列に関連付けられている場合、tableNum には -1が、nameが渡されていれば空の文字列が返されます。

LISTBOX INSERT COLUMN

LISTBOX INSERT COLUMN ({* ;} object ; colPosition ; colName ; colVariable ; headerName ; headerVar {; footerName ; footerVar})

引数	型		説明
*	演算子	⇨	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	⇨	オブジェクト名 (* 指定時) または 変数 (* 省略時)
colPosition	倍長整数	⇨	列の挿入場所
colName	文字	⇨	列オブジェクト名
colVariable	配列, フィールド, 変数	⇨	列配列名 または フィールド または 変数
headerName	文字	⇨	列ヘッダオブジェクト名
headerVar	整数変数	⇨	列ヘッダ変数
footerName	文字	⇨	列フッターオブジェクト名
footerVar	整数変数	⇨	列フッター変数

説明

LISTBOX INSERT COLUMNコマンドは、object引数および * で指定されたリストボックスに列を挿入します。

注: このコマンドは階層モードで表示されているリストボックスの先頭列に適用されてもなにも行いません。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

新しい列は、colPosition引数で指定された列の直前に挿入されます。colPosition引数の値が列の合計数よりも大きい場合、最後の列の後ろにカラムが追加されます。

colNameとcolVariableには、挿入する列のオブジェクト名および変数名を渡します。
・配列タイプのリストボックスの場合、列に表示する内容が格納された配列の名前を渡します。
・セレクションタイプのリストボックスの場合、colVariable引数にはフィールドまたは変数を渡します。結果列の内容は、リストボックスに関連付けられたセレクションのレコードごとに評価されるフィールドまたは変数の値となります。このタイプのコンテンツは、リストボックスの"データソース"プロパティでカレントセレクションまたは命名セレクションを指定した場合にのみ使用できます (リストボックスオブジェクトの管理を参照)。文字列, 数値, 日付, 時間, ピクチャ, ブールタイプのフィールドや変数を使用できます。
セレクションをベースとするリストボックスのコンテキストでは、**LISTBOX INSERT COLUMN**はフィールドや変数などの単純な要素を挿入するために使用できます。フォーミュラやメソッドなどのより複雑な表現式を使用したい場合はLISTBOX INSERT COLUMN FORMULAコマンドを使用します。

注: 同じリストボックス内で、配列をデータソースとする列と、フィールドや変数をデータソースとする列を混在させることはできません。

headerNameとheaderVarには、挿入される列のヘッダのオブジェクト名および変数を渡します。

footerNameとfooterVarにも、挿入される列のヘッダのオブジェクト名および変数を渡すことができます。footerVarを省略した場合、4Dはダイナミック変数を使用します。

注: オブジェクト名は、フォーム内で重複してはいけません。colNameやheaderName、footerNameに渡される名前が既に使用されていないことを確認してください。重複した名前を指定すると、列は作成されず、エラーが生成されます。

例題 1

リストボックスの最後に列を追加します:

```
C_LONGINT(HeaderVarName;$Last;RecNum)
ALL RECORDS([Table1])
$RecNum:=Records in table([Table1])
ARRAY PICTURE(Picture;$RecNum)

$Last:=LISTBOX Get number of columns(*;"ListBox1")+1
LISTBOX INSERT COLUMN(*;"ListBox1";$Last;"ColumnPicture";Picture;"HeaderPicture";HeaderVarName)
```

例題 2

リストボックスの右に列を追加し、[Transport]Fees フィールドの値を関連付けます:

```
$last:=LISTBOX Get number of columns(*;"ListBox1")+1
LISTBOX INSERT COLUMN(*;"ListBox1";$last;"FieldCol";[Transport]Fees;"HeaderName";HeaderVar)
```

LISTBOX INSERT COLUMN FORMULA

LISTBOX INSERT COLUMN FORMULA ({* ;} object ; colPosition ; colName ; formula ; dataType ; headerName ; headerVariable {; footerName ; footerVar})

引数	型		説明
*	演算子	⇨	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	⇨	オブジェクト名 (* 指定時) または 変数 (* 省略時)
colPosition	倍長整数	⇨	列挿入位置
colName	文字	⇨	列オブジェクト名
formula	文字	⇨	列に関連付ける4Dフォーミュラ
dataType	倍長整数	⇨	フォーミュラの結果型
headerName	文字	⇨	列ヘッダーオブジェクト名
headerVariable	整数変数	⇨	列ヘッダー変数
footerName	文字	⇨	列フッターオブジェクト名
footerVar	整数変数	⇨	列フッター変数

説明

LISTBOX INSERT COLUMN FORMULAコマンドは、object引数および * で指定されたリストボックスに列を挿入します。

LISTBOX INSERT COLUMN FORMULAコマンドはLISTBOX INSERT COLUMNコマンドと同様の動作を行いますが、列のコンテンツとしてフォーミュラを指定可能な点が異なります。

このタイプのコンテンツはリストボックスの"データソース"プロパティが**カレントセレクション**または**命名セレクション**に設定されている場合にのみ使用できます (この点に関する詳細はリストボックスオブジェクトの管理を参照してください)。

注: このコマンドは階層モードで表示されているリストボックスの最初の列に適用されてもなにも行いません。

オプションの引数 * を渡すことにより、object引数がオブジェクト名 (文字列) であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

新しい列は、colPosition引数で指定された列の直前に挿入されます。colPosition引数の値が列の合計数よりも大きい場合、最後の列の後ろにカラムが追加されます。

colName 引数には挿入する列のオブジェクト名を渡します。

formula 引数には有効なフォーミュラを渡します。例えば:

- 命令
- フォーミュラエディタで生成したフォーミュラ
- 4D コマンドの呼び出し
- プロジェクトメソッドの呼び出し

コマンドが呼び出される際、formulaは解析・実行されます。

注: フォーミュラが4Dコマンドを呼び出す場合、アプリケーションのローカライズの影響を受けないようにするため、Command nameを使用してください。

dataType 引数は、フォーミュラを実行した結果の型を指定するために使用します。Field and Variable Typesテーマの以下の定数の1つを渡さなければなりません:

定数	型	値
Is Boolean	倍長整数	6
Is Date	倍長整数	4
Is Picture	倍長整数	3
Is Real	倍長整数	1
Is Text	倍長整数	2
Is Time	倍長整数	11

formula の結果が期待するデータ型に対応しない場合、エラーが生成されます。

headerNameとheaderVarには、挿入される列のヘッダーのオブジェクト名および変数を渡します。

footerNameとfooterVarにも、挿入される列のフッターのオブジェクト名および変数を渡すことができます。

注: オブジェクト名は、フォーム内で重複してはいけません。colNameやheaderName、footerNameに渡される名前が既に使用されていないことを確認してください。重複した名前を指定すると、列は作成されず、エラーが生成されます。

例題

リストボックスの右に列を追加し、従業員の年齢を計算するフォーミュラを関連付けます:

```
vAge:="Today's Date-[Employees]BirthDate)\365"
$last:=Get number of listbox columns(*;"ListBox1")+1
LISTBOX INSERT COLUMN FORMULA(*;"ListBox1";$last;"ColFormula";vAge;Is Real;"Age";HeaderVar)
```

LISTBOX INSERT ROWS

LISTBOX INSERT ROWS ({* ;} object ; vPosition {; numRows})

引数	型	説明
*	演算子	⇒指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	⇒オブジェクト名 (* 指定時) または 変数 (* 省略時)
vPosition	倍長整数	⇒行挿入位置
numRows	倍長整数	⇒挿入する行数

説明

LISTBOX INSERT ROWSコマンドは、object引数および * で指定されたリストボックスにひとつ以上の新しい行を挿入します。

注: このコマンドは配列タイプのリストボックスに対してのみ利用できます。セレクションタイプのリストボックスに対して使用した場合、コマンドは何も行わず、OKシステム変数に0が設定されます。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

numRowsが省略されているとデフォルトで行が1つだけ挿入されます。指定された場合、コマンドはこの引数で指定された行数を挿入します。

行はposition 引数で指定された位置に挿入されます。新しい行は、配列タイプや表示状態に関わらず、リストボックスで使用されるすべての配列のこの位置に自動で挿入されます。

position の値がリストボックスの総行数より大きい場合、行は各配列の一番最後に追加されます。0の場合、行は各配列の先頭に追加されます。負数の場合、何も行いません。

LISTBOX MOVE COLUMN

LISTBOX MOVE COLUMN ({* ;} object ; colPosition)

引数	型		説明
*	演算子	→	指定時: objectはオブジェクト名 (文字) 省略時: objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
colPosition	倍長整数	→	列の新しい位置

説明

LISTBOX MOVE COLUMNコマンドは、objectと * によって指定された列を、プログラムによって実行中のフォームにおいて移動させます。(アプリケーションモード)デザインモードで生成されたオリジナルのフォームは変更されません。

objectと * 演算子を使用して移動する列を指定します。任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数で列変数を指定します。

指定した列は、colPosition 引数で指定した位置の一つ前に移動されます。colPosition が列全体の総数より大きい場合、指定した列は最後の列の後ろに移動されます。

Note: このコマンドは階層リストボックスの最初の列を指定した場合には何もしません。

このコマンドには列が「ドラッグしない列数」や「スクロールしない列数」で指定されているかどうかが関わってきます。つまり、例えば「ドラッグしない列数」に含まれる場合は列を移動させることはできません。

この機能は以前の4Dからアプリケーションモードには存在するものでした。ユーザーは「ドラッグしない列数」に含まれない列であればマウスで動かすことができました。しかしながら、ユーザーによって列が移動された場合と異なり、このコマンドは On Column Moved イベントを生成しません。

例題

リストボックスの2列目と3列目を入れ替えたい場合:

```
LISTBOX MOVE COLUMN(*;"column2";3)
```

LISTBOX MOVED COLUMN NUMBER

LISTBOX MOVED COLUMN NUMBER ({* ;} object ; oldPosition ; newPosition)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
oldPosition	倍長整数	←	移動された列の前の位置
newPosition	倍長整数	←	移動された列の新しい位置

説明

LISTBOX MOVED COLUMN NUMBER コマンドは、引数objectおよび * で指定されたリストボックス内で移動された列の以前の位置oldPositionと新しい位置newPositionを返します。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はの節を参照してください。

このコマンドはOn column movedフォームイベントで使用します (Form event コマンド参照)。

Note: このコマンドは、非表示の列を考慮します。

LISTBOX MOVED ROW NUMBER

LISTBOX MOVED ROW NUMBER ({* ;} object ; oldPosition ; newPosition)

引数	型	説明
*	演算子	⇒指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	⇒オブジェクト名 (* 指定時) または 変数 (* 省略時)
oldPosition	倍長整数	⇐移動された行の以前の位置
newPosition	倍長整数	⇐移動された行の新しい位置

説明

LISTBOX MOVED ROW NUMBER コマンドは、引数objectおよび * で指定されたリストボックス内で移動された行の以前の位置oldPositionと新しい位置newPositionを返します。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はの節を参照してください。

このコマンドはOn row movedフォームイベントで使用します (Form event コマンド参照)。

Note: このコマンドは、非表示の行を考慮しません。

LISTBOX SELECT BREAK

LISTBOX SELECT BREAK ({* ;} object ; row ; column {; action})

引数	型		説明
*	演算子	⇒	指定時、objectはオブジェクト名 (文字列) 省略時、objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または 変数 (* 省略時)
row	倍長整数	⇒	ブレーク行の番号
column	倍長整数	⇒	ブレーク列の番号
action	倍長整数	⇒	選択アクション

説明

LISTBOX SELECT BREAK を使用して objectと*で指定したリストボックス中でブレーク行を選択できます。リストボックスは階層モードで表示されていなければなりません。

オプションの* 引数を渡した場合、object引数はオブジェクト名 (文字列) です。この引数を渡さない場合objectは変数です。この場合、文字列ではなく変数参照を渡します。

ブレーク行は階層を表現するために追加されますが、それは配列の既存の行には対応しません。選択するためにブレーク行を指定するには、rowおよびcolumn引数に、対応する配列中の最初のオカレンスに対応する行と列の番号を渡さなければなりません。これらの値はユーザがブレーク行を選択したとき、LISTBOX GET CELL POSITIONコマンドから返されます。この原則は階層リストボックスの管理の"選択や位置の管理" で説明されています。

action引数が渡されると、ブレーク行が既にリストボックス中に存在するときの実行する選択アクションを設定できます。値または"List Box"テーマの以下の定数を渡すことができます:

定数	型	値	コメント
Add to listbox selection	倍長整数	1	選択された行は既存の選択行に追加されます。指定した行が既存の選択に属している場合、コマンドは何も行いません。
Remove from listbox selection	倍長整数	2	指定された行は既存の選択行から取り除かれます。指定した行が既存の選択に属さない場合、コマンドは何も行いません。
Replace listbox selection	倍長整数	0	選択された行が新しい選択行となり、既存のものと置き換えられます。このコマンドは、ユーザが行をクリックした場合と同じ結果になります。これは (action 引数が省略された時の) デフォルトの動作です。

例題

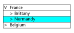
リストボックスに表示されている以下の配列があります:

(T1)	(T2)	(T3)	(T4)
France	Brittany	Brest	120000
France	Brittany	Quimper	80000
France	Brittany	Rennes	200000
France	Normandy	Caen	220000
France	Normandy	Deauville	4000
France	Normandy	Cherbourg	41000
Belgium	Wallonia	Namur	111000
Belgium	Wallonia	Liege	200000
Belgium	Flanders	Antwerp	472000
Belgium	Flanders	Louvain	95000

"Normandy"ブレーク行を選択します:

```
$row:=Find in array(T2;"Normandy")
$column:=2
LISTBOX COLLAPSE(*;"MyListbox") // 全レベルを折りたたむ
LISTBOX SELECT BREAK(*;"MyListbox";$row;$column)
```

以下のような結果になります:



LISTBOX SELECT ROW

LISTBOX SELECT ROW ({* ;} object ; vPosition {; action})

引数	型		説明
*	演算子	➡	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	➡	オブジェクト名 (* 指定時) または 変数 (* 省略時)
vPosition	倍長整数	➡	選択する行番号
action	倍長整数	➡	選択アクション

説明

LISTBOX SELECT ROW コマンドは、object引数および * で指定されたリストボックス内において、positionに渡した番号の行を選択します。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

オプション引数actionを指定すると、行の選択がリストボックス内に既に存在している場合に実行する選択アクションを指定できます。この引数には値または次の定数のいずれかを渡すことができます（List Boxテーマ）:

定数	型	値	コメント
Listbox add to selection	倍長整数	1	選択された行は既存の選択行に追加されます。指定した行が既存の選択に属している場合、コマンドは何も行いません。
Listbox remove from selection	倍長整数	2	指定された行は既存の選択行から取り除かれます。指定した行が既存の選択に属さない場合、コマンドは何も行いません。
Listbox replace selection	倍長整数	0	選択された行が新しい選択行となり、既存のものと置き換えられます。このコマンドは、ユーザが行をクリックした場合と同じ結果になります。これは (action 引数が省略された時の) デフォルトの動作です。

position引数が既存の行番号と一致しない時、コマンドは以下のように動作します:

- positionが0より小さい場合、actionの値に関係なく、何もしません。
- positionが0かつ、actionの値がListbox replace selectionまたは省略された場合、リストボックスのすべての行が選択されます。actionの値がListbox remove from selectionの場合、リストボックスのすべての行の選択が解除されます。
- positionがリストボックスに含まれるすべての行数より大きい場合 (配列タイプのリストボックスの場合のみ)、リストボックスと関連づけられているブール配列が自動的にリサイズされ、選択アクションが行われます。このメカニズムは、**LISTBOX SELECT ROW** を、リストボックスで即座の同期を起こさない"標準"の配列管理コマンド (APPEND TO ARRAYなど) とともに使用できることを意味します。

メソッドの実行後、配列は同期されます。リストボックスのソース配列が実際にリサイズされたならば、選択アクションは実行されます。そうでなければリストボックスと関連づけられたブールの配列はその初期のサイズに戻り、コマンドは何もしません。

Notes:

- 選択した列をリストボックスに表示するために、自動でスクロールしたい場合、OBJECT SET SCROLL POSITIONコマンドを使います。
- 列を編集モードに切り換えるには、EDIT ITEMコマンドを使います。
- position に渡された数値が、リストボックス中で非表示の行に該当する場合、行は選択されますが表示されません。

LISTBOX SET ARRAY

LISTBOX SET ARRAY ({* ;} object ; arrType ; arrPtr)

引数	型		説明
*	演算子	➡	指定時:objectはオブジェクト名(文字列)省略時:objectは変数
object	フォームオブジェクト	➡	オブジェクト名(* 指定時)、または変数(* 省略時)
arrType	倍長整数	➡	配列のタイプ
arrPtr	ポインター	➡	プロパティに関連付ける配列を指定

説明

注: このコマンドは配列型のリストボックスに対してのみ有効です。

LISTBOX SET ARRAYコマンドは、object and * によって指定されたリストボックスもしくはリストボックスコラムに、スタイル配列またはカラー配列を関連付けます。

注: これまでの 4D で配列型のリストボックスにスタイル、文字色、背景色の配列を関連付けるためには、デザインモードのプロパティリストを使用するしかありませんでした。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数で変数を指定します。対象がリストボックスなのか列なのかを、object 引数で指定します。

arrType 引数にはリストボックスまたは列に関連付けたい配列の種類を、"List Box" のテーマ内にある以下の定数を渡すことによって指定します。

定数	型	値
Listbox background color array	倍長整数	1
Listbox font color array	倍長整数	0
Listbox style array	倍長整数	2

arrPtr 引数には、制御したいプロパティを制御するための配列に対するポインターを渡します(フォントカラー、背景色あるいはフォントスタイル)。

例題

4列目のフォントカラー配列を10列目にも使いたいという場合を考えます。

```
  // 4列目で使用している配列に対するポインターを取得します。
$Pointer:=LISTBOX Get array(*;"Col4";Listbox font color array)
  // 存在するかどうかをチェックします
If(Not(Nil($Pointer)))
  // 10列目へ適用します。
   LISTBOX SET ARRAY(*;"Col10";Listbox font color array;$Pointer)
End if
```

LISTBOX SET COLUMN FORMULA

LISTBOX SET COLUMN FORMULA ({* ;} object ; formula ; dataType)

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列)、省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
formula	文字	→	列に割り当てる4Dフォーミュラ
dataType	倍長整数	→	フォーミュラの結果型

説明

LISTBOX SET COLUMN FORMULAコマンドはobjectと* 引数で指定したリストボックス列に割り当てられたformulaを変更します。フォーミュラはリストボックスプロパティのデータソースが**カレントセレクション**あるいは**命名セレクション**の場合のみ使用できます。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。この引数はリストボックス列を指定しなければなりません。

formula には以下のような、有効な式を指定できます:

- インストラクション
- フォーミュラエディターで生成したフォーミュラ
- 4Dコマンドの呼び出し
- プロジェクトメソッドの呼び出し

コマンドが呼び出されるとフォーミュラが解析され実行されます。

注: アプリケーションランゲージに左右されない、4Dコマンドを呼び出すフォーミュラを指定するためにはCommand nameを使用します。

dataType 引数にはフォーミュラ実行時に返されるデータの型を指定します。この引数にはField and Variable Typesテーマ内の定数を渡します。フォーミュラの結果が期待するデータ型に一致しない場合、エラーが生成されます。

LISTBOX SET COLUMN WIDTH

LISTBOX SET COLUMN WIDTH ({* ;} object ; width {; minWidth {; maxWidth}})

引数	型	説明
*	演算子	→ 指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→ オブジェクト名 (* 指定時) または 変数 (* 省略時)
width	倍長整数	→ 列幅 (ピクセル単位)
minWidth	倍長整数	→ 列の最小幅 (ピクセル単位)
maxWidth	倍長整数	→ 列の最大幅 (ピクセル単位)

説明

SET LISTBOX COLUMN WIDTH コマンドを使用し、object引数および * で指定されたオブジェクト（リストボックス、列、またはヘッダ）の任意の列の幅、またはすべての列の幅をプログラムから変更することができます。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

width引数には、オブジェクトの新しい幅（ピクセル単位）を渡します。

objectがリストボックスオブジェクトである場合、リストボックスのすべての列サイズが変更されます。

objectが列または列ヘッダである場合、その列のサイズだけが変更されます。

オプションのminWidthとmaxWidth引数を使用して、列の手動によるサイズ変更の制限を設定できます。minWidthとmaxWidth引数にそれぞれピクセル単位で最小および最大幅を渡すことができます。ユーザにサイズ変更をさせたくない場合は、width、minWidthおよびmaxWidthに同じ値を渡します。

LISTBOX SET FOOTER CALCULATION

LISTBOX SET FOOTER CALCULATION ({* ;} object ; calculation)

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列)、省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
calculation	倍長整数	→	フッターエリアの計算タイプ

説明

LISTBOX SET FOOTER CALCULATIONコマンドはobjectと * 引数で指定したリストボックスのフッターに割り当てる自動計算を設定します。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。

object 引数では以下を指定できます:

- フッターエリアの変数あるいはオブジェクト名。この場合コマンドはこのエリアに適用されます。
- リストボックス列の変数あるいはオブジェクト名。この場合コマンドはこの列のフッターエリアに適用されます。
- リストボックスの変数あるいはオブジェクト名。この場合コマンドはリストボックスのすべてのフッターエリアに適用されます。

calculation 引数には実行する計算式を指定するListbox Footer Calculationテーマの定数を渡します:

定数	型	値	コメント
Listbox footer average	倍長整数	6	数値または時間型の列に使用します。 結果のデフォルト型: 実数
Listbox footer count	倍長整数	5	数値、テキスト、日付、時間、ブール、またはピクチャー型の列に使用します。 結果のデフォルト型: 倍長整数
Listbox footer custom	倍長整数	1	4Dは計算を行いません。フッター変数はプログラムを使用して計算されなければなりません。 結果のデフォルト型: フッター変数に指定された型
Listbox footer max	倍長整数	3	数値、日付、時間、またはブール型の列に使用します。 結果のデフォルト型: 列配列またはフィールドに対応する型
Listbox footer min	倍長整数	2	数値、日付、時間、またはブール型の列に使用します。 結果のデフォルト型: 列配列またはフィールドに対応する型
Listbox footer std deviation	倍長整数	7	数値または時間型の列に使用します (配列タイプのリストボックスのみ)。 結果のデフォルト型: 実数
Listbox footer sum	倍長整数	4	数値、ブールまたは時間型の列に使用します。 結果のデフォルト型: 列配列またはフィールドに対応する型
Listbox footer sum squares	倍長整数	9	数値または時間型の列に使用します (配列タイプのリストボックスのみ)。 結果のデフォルト型: 実数
Listbox footer variance	倍長整数	8	数値または時間型の列に使用します (配列タイプのリストボックスのみ)。 結果のデフォルト型: 実数

定義済み計算式では列のすべての値が計算対象になることに留意してください。これには非表示行も含まれます。表示されている行に計算対象を限定したい場合は Listbox footer custom 定数を使用し、計算をカスタマイズしなければなりません。

列のデータ型や (objectにリストボックスを指定した場合) リストボックス中の1つの列でも設定したcalculationと互換性がない場合、フッターは変更されずエラー18が生成されます。列にフォーミュラが設定されている場合 (セレクションタイプのリストボックス)、エラー10が生成されます。

注: フッターエリア変数の型は (コードで明示的に型宣言されていない場合)、プロパティリストで設定された計算に基づき自動で型付けされます (リストボックスフッター特有のプロパティ参照)。変数型が、**LISTBOX SET FOOTER CALCULATION**コマンドが期待する結果の型に対応しない場合、型エラーが生成されます。例えば列が日付を表示しているとき、フッターの計算式が '最大値' であれば、フッターエリアの変数は日付型になります。このとき **LISTBOX SET FOOTER CALCULATION**(footer;Listbox footer count) を実行すると、変数型日付に対して式の結果が倍長整数になるため、エラーが生成されます。

LISTBOX SET FOOTERS HEIGHT

LISTBOX SET FOOTERS HEIGHT ({* ;} object ; height {; unit})

引数	型		説明
*	演算子	⇒	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または変数 (* 省略時)
height	倍長整数	⇒	行の高さ
unit	倍長整数	⇒	高さを指定する単位: 0または省略時 = ピクセル、1 = 行

説明

LISTBOX SET FOOTERS HEIGHTコマンドはobjectと* 引数で指定したリストボックスのフッター行の高さを変更します。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。リストボックスあるいはリストボックスフッターを指定できます。

設定する高さを height 引数に渡します。unit 引数を省略するとデフォルトで単にはピクセルです。単位を変更するにはList Boxテーマの定数を unit 引数に渡します:

定数	型	値	コメント
Listbox lines	倍長整数	1	高さを行数で指定。4Dはフォント設定に応じて高さを計算します。
Listbox pixels	倍長整数	0	高さをピクセルで指定 (デフォルト)。

注: 行の高さの計算に関する詳細はデザインリファレンスマニュアルを参照してください。

LISTBOX SET GRID

LISTBOX SET GRID ({* ;} object ; horizontal ; vertical)

引数	型		説明
*	演算子	⇒	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または 変数 (* 省略時)
horizontal	ブール	⇒	True = 表示, False = 非表示
vertical	ブール	⇒	True = 表示, False = 非表示

説明

LISTBOX SET GRIDコマンドを使用し、object引数および * で指定されたリストボックスのグリッドを構成する、横および縦グリッドラインを表示、または非表示に設定することができます。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

horizontalとverticalには、対応するグリッドラインを表示するか（**True**）、表示しないか（**False**）を示すブール値を渡します。デフォルトでは、グリッドが表示されます。

LISTBOX SET GRID COLOR

LISTBOX SET GRID COLOR ({* ;} object ; color ; horizontal ; vertical)

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
color	倍長整数	→	RGBカラー値
horizontal	ブール	→	水平グリッドラインにカラーを適用
vertical	ブール	→	垂直グリッドラインにカラーを適用

説明

SET LISTBOX GRID COLOR コマンドを使用し、object引数および * で指定されたリストボックスオブジェクト上のグリッドの色を変更することができます。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はの節を参照してください。

colorには、RGBカラーの値を渡します。RGBカラーに関する詳細は、SET RGB COLORSコマンドの説明を参照してください。

horizontalとverticalを使用し、カラーを適用するグリッドラインを指定することができます:

- horizontalにTrueを渡した場合、水平グリッドラインにカラーが適用されます。Falseを渡すと、カラーは変更されません。
- verticalにTrueを渡した場合、垂直グリッドラインにカラーが適用されます。Falseを渡すと、カラーは変更されません。

LISTBOX SET HEADERS HEIGHT

LISTBOX SET HEADERS HEIGHT ({* ;} object ; height {; unit})

引数	型		説明
*	演算子	⇨	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	⇨	オブジェクト名 (* 指定時) または変数 (* 省略時)
height	倍長整数	⇨	ヘッダーの高さ
unit	倍長整数	⇨	高さを指定する単位: 0または省略時 = ピクセル、1 = 行

説明

LISTBOX SET HEADERS HEIGHTコマンドはobjectと* 引数で指定したリストボックスのヘッダー行の高さを変更します。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。

リストボックスあるいはリストボックスヘッダーのいずれかを指定できます。

設定する高さを height 引数に渡します。unit 引数を省略するとデフォルトで高さはピクセル単位であらわされます。単位を変更するには unit 引数にList Boxテーマの以下の定数を渡します:

定数	型	値	コメント
Listbox lines	倍長整数	1	高さを行数で指定。4Dはフォント設定に応じて高さを計算します。
Listbox pixels	倍長整数	0	高さをピクセルで指定 (デフォルト)。

ヘッダー高さの最小値はシステムに設定された値を使用します。

これはWindowsで24ピクセル、Mac OSで17ピクセルです。

height引数にこれより小さな値を渡すと、最小値が適用されます。

注: 行の高さの計算についてはデザインリファレンスマニュアルを参照してください。

LISTBOX SET HIERARCHY

LISTBOX SET HIERARCHY ({* ;} object ; hierarchical {; hierarchy})

引数	型		説明
*	演算子	→	指定時、objectはオブジェクト名 (文字列) 省略時、objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
hierarchical	ブール	→	True = 階層リストボックス False = 非階層リストボックス
hierarchy	ポインター配列	→	ポインタの配列

説明

LISTBOX SET HIERARCHY コマンドを使用してobjectと*で指定されたリストボックスを階層モードにするか非階層モードにするか設定できます。.

Note:このコマンドは配列を表示するリストボックスに対してのみ機能します。セレクション表示モードのリストボックスに対してこのコマンドが適用された場合、なにも行いません。

オプションの* 引数を渡した場合、object引数はオブジェクト名 (文字列) です。この引数を渡さない場合objectは変数です。この場合、文字列ではなく変数参照を渡します。

ブール型のhierarchical引数を使用してリストボックスのモードを指定します:

- Trueを渡すと、リストボックスは階層モードで表示されます。
- Falseを渡すと、リストボックスは (階層モードではない) 標準配列モードで表示されます。

階層モードのリストボックスを渡すと、特定のプロパティは自動で制限されます。詳しくは階層リストボックスの管理を参照してください。

hierarchy引数には、階層を構成するために使用されるリストボックスの配列を指定します (例題参照)。リストボックスを階層モードで表示し、この引数を省略すると:

- すでにリストボックスが階層モードの場合、コマンドはなにも行いません。
- リストボックスが現在非階層モードで、過去にも階層であると設定されたことがなければ、デフォルトで最初の配列が階層として使用されます。
- リストボックスが現在非階層モードで、以前に階層モードとして設定されたことがあれば、最後の時点の階層が再構築されます。

例題

aCountry、aRegion、そしてaCity配列をリストボックスの階層として定義する:

```
ARRAY POINTER($ArrHierarch;3)
$ArrHierarch{1}:=->aCountry // 一番目のブレークレベル
$ArrHierarch{2}:=->aRegion // 二番目のブレークレベル
$ArrHierarch{3}:=->aCity // 三番目のブレークレベル
LISTBOX SET HIERARCHY(*;"mylistbox";True;$ArrHierarch)
```

LISTBOX SET LOCKED COLUMNS

LISTBOX SET LOCKED COLUMNS ({* ;} object ; numColumns)

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または変数 (* 省略時)
numColumns	倍長整数	→	横スクロールしない列の数

説明

LISTBOX SET LOCKED COLUMNSコマンドはobject と* 引数で指定したリストボックスで左端からはじめてnumColumns列をロックします。

ロックされた列はリストボックスの左側に表示され、スクロールされることはありません。詳細はデザインリファレンスマニュアルを参照してください。

オプションの * 引数を渡した場合、object 引数はオブジェクト名 (文字列) です。この引数を渡さない場合 object は変数です。この場合文字列ではなく変数参照を渡します。

numColumns には1からリストボックスの総列数-1までの値を渡せます。X列を持つリストボックスの場合、X-1より大きな値を渡すと、自動でX-1が設定されます。

列のロックを解除するにはnumColumnsに0を渡します。

LISTBOX SET ROW COLOR

LISTBOX SET ROW COLOR ({* ;} object ; row ; color {; colorType})

引数	型		説明
*	演算子	→	指定時:objectはオブジェクト名(文字列)省略時:objectは変数
object	フォームオブジェクト	→	オブジェクト名(*指定時)、または変数(*省略時)
row	倍長整数	→	列番号
color	倍長整数	→	RGBカラー
colorType	倍長整数	→	Listbox font color(デフォルトの値) またはListbox background color

説明

注: このコマンドは配列型のリストボックスに対してのみ有効です。

LISTBOX SET ROW COLORコマンドは、objectと * によって指定された配列リストボックスの中の、行またはセルの色を指定します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数で列変数を指定します。

対象がリストボックスなのか列なのかを、object 引数で指定します。

- object がリストボックスを指定するとき、コマンドはその中で指定した行全体に反映されます。
- object がリストボックスの列を指定するとき、コマンドはその列内の、指定された行にある単一のセルに対して反映されます。

row 引数には、新しい色を反映させたい列を指定する番号を渡します。

注: このコマンドは列の表示/非表示の状態は無視します。

color 引数には、指定したい色をRGBカラー値で渡します。RGBカラーについての詳細は、OBJECT SET RGB COLORS コマンドの詳細を参照して下さい。一つ上の行の配色をそのまま使用したい場合には、color 引数 に Listbox inherited 定数を渡します。

colorType 引数には、行の背景色を指定したいのかフォントカラーを指定したのかによって、Listbox background color または Listbox font color のどちらかの定数を渡します。省略時には、指定した色はフォントカラーとして使用されます。

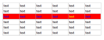
このコマンドは、列またはリストボックスに対して定義されていた可能性のあるカラー配列内の値を変更します。この配列が未定義だった場合、このコマンドは、LISTBOX Get array コマンドを使用してアクセス可能な配列を動的に作成します。

他のリストボックスプロパティ(全体のプロパティ、列のカラー配列など)によって、このコマンドと異なる色が指定された場合、4D内の優先順位に応じて処理されます。この優先順位の詳細に関しては、*Design Reference* を参照して下さい。

例題

配列型のリストボックス内において、行全体と、行内の単一のセルの色を指定したい場合:

```
  // 単一のセルのフォントカラー（黄色）の定義
 LISTBOX SET ROW COLOR(*;"Col5";3;0x00FFFF00)

  // 3行目の背景色とフォントカラーの定義
  // 背景色は赤、フォントカラーは青
 LISTBOX SET ROW COLOR(*;"ListBox";3;0x00FF0000;Listbox background color)
 LISTBOX SET ROW COLOR(*;"List Box";3;0x000000FF)
```



text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

LISTBOX SET ROW FONT STYLE

LISTBOX SET ROW FONT STYLE ({* ;} object ; row ; style)

引数	型		説明
*	演算子	→	指定時:objectはオブジェクト名(文字列)省略時:objectは変数
object	フォームオブジェクト	→	オブジェクト名(*指定時)、または変数(*省略時)
row	倍長整数	→	列番号
style	倍長整数	→	フォントスタイル

説明

注: このコマンドは配列型のリストボックスに対してのみ有効です。

LISTBOX SET ROW FONT STYLEコマンドは、object and * によって指定された配列型リストボックスの、行またはセルのフォントスタイルを設定します。

任意の * 演算子を渡した場合、object 引数でオブジェクト名を文字列で指定します。省略時には object 引数で列変数を指定します。

対象がリストボックスなのか列なのかを、object 引数で指定します。

- object がリストボックスを指定するとき、コマンドはその中で指定した行全体に反映されます。
- object がリストボックスの列を指定するとき、コマンドはその列内の、指定された行にある単一のセルに対して反映されます。

row 引数には、新しいスタイルを反映させたい列を指定する番号を渡します。

注: このコマンドは列の表示/非表示の状態は無視します。

style 引数には、指定したいスタイルの値を渡します。指定する際には Font Styles テーマ内にある、以下の定数のどれか一つ、あるいはその組み合わせを指定して下さい。

定数	型	値
Bold	倍長整数	1
Italic	倍長整数	2
Plain	倍長整数	0
Underline	倍長整数	4

リストボックスや列にフォントスタイル配列が設定されている場合、指定された列の要素に関してのみ変更されます。言い換えると、この場合コマンドを実行するのはフォントスタイル配列の要素を変更するのと同等の効果があります。
リストボックスや列にフォントスタイル配列が何も設定されていない場合、このコマンドが使用されたときに作成されます。この配列にはLISTBOX Get array コマンドを使用することによってアクセス可能です。

他のリストボックスプロパティ(一般のプロパティ、行スタイル配列など)によって、このコマンドと異なるフォントスタイルが指定された場合、4D内の優先順位に応じて処理されます。この優先順位の詳細に関しては、*Design Reference* マニュアルを参照して下さい。

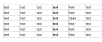
注: リストボックス全体に対するスタイルの指定より単一の列に対するスタイルの方が優先されるので、リストボックス全体に対してこのコマンドは、列に対するスタイル配列が設定されていない場合のみ有効です。

例題

ある配列リストボックスに以下のスタイルを適用する場合を考えます。

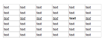
- フォントスタイル配列がリストボックス全体に適用されているとします。(*ArrGlobalStyle*)
- フォントスタイル配列が、5行目に適用されているとします。(*ArrCol5Style*)
- 他の列にはスタイル配列は適用されていません。

```
LISTBOX SET ROW FONT STYLE(*;"Col5";3;Bold)
 // ArrCol5Style{3}:=Bold と同様の効果
```



text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	**text**	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

```
LISTBOX SET ROW FONT STYLE(*;"List Box";3;Italic+Underline)
 // ArrGlobalStyle{3}:=Italic+Underline と同様の効果
```



text	text	text	text	text	text
text	text	text	text	text	text
text	*text*	*text*	*text*	**text**	*text*
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

2つ目のコマンドの後、3行目のセルは下線付きのイタリックになりましたが、5列目のセルだけは太字のままになっています(列スタイル配列の方がリストボックス配列より優先されるからです)。

LISTBOX SET ROWS HEIGHT

LISTBOX SET ROWS HEIGHT ({* ;} object ; height {; unit})

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
height	倍長整数	→	行の高さ
unit	倍長整数	→	高さを表す単位: 0または省略時はピクセル、1の場合行単位

説明

LISTBOX SET ROWS HEIGHTコマンドを使用すると、object引数および * で指定されたリストボックス内の行の高さをプログラムで変更することができます。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

unit 引数を省略するとデフォルトで高さはピクセル単位で返されます。単位を変更するには unit 引数にList Boxテーマの以下の定数を渡します:

定数	型	値	コメント
Listbox lines	倍長整数	1	高さを行数で指定。4Dはフォント設定に応じて高さを計算します。
Listbox pixels	倍長整数	0	高さをピクセルで指定 (デフォルト)。

注: 行の高さの計算に関する詳細はデザインリファレンスを参照してください。

LISTBOX SET STATIC COLUMNS

LISTBOX SET STATIC COLUMNS ({* ;} object ; numColumn)

引数	型		説明
*	演算子	⇒	指定時objectはオブジェクト名 (文字列) 省略時objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または変数 (* 省略時)
numColumn	倍長整数	⇒	ドラッグで移動しない列数

説明

LISTBOX SET STATIC COLUMNSコマンドはobjectと * 引数で指定したリストボックス中のスタティック列の数を左端からnumColumnsに設定します。

リストボックス内でスタティック列は移動することができません。

注: スタティック列とロックされた列は異なる機能です。詳細はランゲージリファレンスマニュアルを参照してください。

LISTBOX SET TABLE SOURCE

LISTBOX SET TABLE SOURCE ({* ;} object ; tableNum | name {; highlightName})

引数	型		説明
*	演算子	→	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
tableNum | name	倍長整数, 文字	→	カレントセレクションが使用されるテーブル番号 または使用される命名セレクション
highlightName	文字	→	ハイライトセットの名前

説明

LISTBOX SET TABLE SOURCEコマンドは、* と object 引数で指定されるリストボックスに表示されるデータのソースを変更するために使用します。

注: このコマンドは、リストボックスの"データソース"プロパティが**カレントセレクション**または**命名セレクション**に設定されている場合にのみ使用できます(詳細はリストボックスオブジェクトの管理を参照してください)。配列タイプのリストボックスにこのコマンドを適用しても何も行いません。

オプションの引数 * を渡すことにより、object引数がオブジェクト名 (文字列) であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はオブジェクトプロパティを参照してください。

tableNum 引数としてテーブル番号を渡すと、リストボックスにはそのテーブルのカレントセレクションが表示されます。

name 引数として命名セレクション名を渡すと、リストボックスには命名セレクションに属するデータが表示されます。

オプションの highlightName 引数はリストボックスにハイライトセットを割り当てます。ハイライトセットはリストボックス中でユーザーがハイライトしたレコードを管理します。

リストボックスに列が既に含まれている場合、コマンド実行後に内容が更新されます。

注: 最適化のため、このコマンドは非同期で処理されます。言い換えればリストボックスのソースはコマンドが呼び出されたメソッドの実行が完全に終了したときに変更されます。

LISTBOX SORT COLUMNS

LISTBOX SORT COLUMNS ({* ;} object ; colNum ; order {; colNum2 ; order2 ; ... ; colNumN ; orderN})

引数	型		説明
*	演算子	⇒	指定時, objectはオブジェクト名 (文字列) 省略時, objectは変数
object	フォームオブジェクト	⇒	オブジェクト名 (* 指定時) または 変数 (* 省略時)
colNum	倍長整数	⇒	並び替える列番号
order	演算子	⇒	">": 昇順ソート または "<": 降順ソート

説明

LISTBOX SORT COLUMNSコマンドは、1つ以上の列の値に基づいて、object引数および*で指定されたリストボックスの行を並べ替えます。

オプションの引数 * を渡すことにより、object引数がオブジェクト名（文字列）であることを示します。この引数を渡さない場合、object引数が変数であることを示します。この場合、文字列ではなく変数参照を指定します。オブジェクト名についての詳細はの節を参照してください。

colNumには、ソート条件として使用したい列番号を指定します。ピクチャタイプとポインタタイプを除き、任意のタイプの配列データを使用することができます。

orderには、並び替え順を表わす>または<記号を渡します。orderに>を指定すると、並び替えは昇順になります。orderに<を指定すると、並び替えは降順になります。

マルチレベルソートを定義することができます。これを行うには、必要なだけcolNum;orderのペアを渡します。並び替えレベルは、コマンド内の引数の位置によって決まります。

リストボックス操作の原則に従い、それぞれの列は同期化されます。つまり列の並び順は、自動的にそのオブジェクトの他のすべての列に受け継がれます。

リソース

- リソース
- CLOSE RESOURCE FILE
- GET ICON RESOURCE
- Get indexed string
- GET PICTURE RESOURCE
- GET RESOURCE
- Get resource name
- Get resource properties
- Get string resource
- Get text resource
- Open resource file
- RESOURCE LIST
- RESOURCE TYPE LIST
- STRING LIST TO ARRAY
- ARRAY TO STRING LIST
- Create resource file
- DELETE RESOURCE
- Get component resource ID
- SET PICTURE RESOURCE
- SET RESOURCE
- SET RESOURCE NAME
- SET RESOURCE PROPERTIES
- SET STRING RESOURCE
- SET TEXT RESOURCE

リソース

Compatibility notes about writing resources (4D v13)

4D v11 SQLリリース時にアナウンスされていた通り、Macリソースメカニズムの利用は廃止されました。**4D v13より、"リソース"テーマ内のリソースファイル書き込みコマンドは無効となり、動作しなくなります。**リソースから読み込みを行うコマンドのみ互換性のため現時点で利用可能です。

リソース管理メカニズムに関する互換性の注意 (4D v11)

バージョン11よりリソースの管理が変更されました。Apple社より示された方針や最近のMac OSの実装に沿い、厳密な意味でのリソースのコンセプト (下記の定義参照) は廃止され、徐々に機能が取り除かれます。リソースにより提供されていたニーズをサポートするために、文字列の翻訳に使用するXLIFFファイルや.png ピクチャなど、新しいメカニズムが実装されました。実際、リソースファイルは標準のファイルに置き換えられます。4Dはこの進化をサポートし、バージョン11より、既存のシステムとの互換性を保持しつつ、データベースの翻訳を管理する新しいツールを提供します。

互換性

互換性を保持しつつ、既存のアプリケーションを徐々に新しい方式に移行できるようにするため、4D v11では以前のリソースメカニズムが引き続き動作します。以下の点において異なる点に留意してください:

- 4Dはリソースファイルとリソースチェーン (複数のリソースファイルを開く) をサポートします。リソースチェーンには特に変換されたデータベースにおいて.rsrや.4drファイル (自動で開かれます) と、このテーマのコマンドを使用して開かれたカスタムリソースが含まれます。
- しかし、内部的なアーキテクチャの進化のため、このテーマのコマンドや動的参照を使用して直接4Dやシステムのリソースにアクセスすることはできなくなりました。開発者の中にはインタフェースに使用する目的で4Dの内部 リソースを使用している場合があります。(例えば付きの名前やコマンド名を含むリソース)。この方法は完全に禁止されます。ほとんどの場合、4D内部リソースの代わりに他の方法を使用できます (定数、ランゲージコマンド等)。既存データベースの更新に対するインパクトを緩和するために、しばしば使用されるリソースを外部ファイル化するという代わりのシステムが実装されました。しかしながら変換されたデータベースにおいて、4D内部リソースを呼び出す部分を取り除くことを強くお勧めします。
- バージョン11より、4Dで作成されたデータベースには.RSR (ストラクチャリソース) と .4DR (データリソース) ファイルは含まれません。

新しいリソース管理の原則

4D v11では、"リソース"という用語は"アプリケーションインターフェースのために翻訳が必要なファイル"と理解する必要があります。リソースの新しいアーキテクチャは"Resources"という名前のフォルダーに基づき、このフォルダーはデータベースのストラクチャーファイル (.4db or .4dc) と同階層になければなりません。このフォルダールダには、インターフェースの翻訳やカスタマイズに使用するファイル (ピクチャーファイル, テキストファイル, XLIFFファイル等) を置きます。
このフォルダーには前世代のリソースファイル (.rsr files) を置くこともできます。このリソースファイルは自動ではリソースチェーンに含められないことに留意してください。4D標準のリソース処理コマンドを使用して開かなければなりません。4Dはこのフォルダーを処理する際に、特にXLIFFファイルの管理において自動のメカニズムを適用します (このポイントは4D Design Referenceマニュアルで説明しています)。"リソース"テーマの2つのコマンド (Get indexed stringとSTRING LIST TO ARRAYコマンド) がこの新しいアーキテクチャをサポートしています。

CLOSE RESOURCE FILE

CLOSE RESOURCE FILE (resFile)

引数	型		説明
resFile	DocRef	→	リソースファイル参照番号

説明

CLOSE RESOURCE FILEコマンドは、引数resFileに渡された参照番号を持つリソースファイルを閉じます。

何度も同じリソースファイルを開いても、そのリソースファイルを閉じるには**CLOSE RESOURCE FILE**コマンドを1回呼び出すだけで済みます。

4Dアプリケーションやデータベースリソースファイルに**CLOSE RESOURCE FILE**コマンドを適用しても、このコマンドはそれを検知し何も行いません。

無効なリソースファイル参照番号を渡すと、このコマンドは何も行いません。

Open resource fileを使って開いたリソースファイルに対し、**CLOSE RESOURCE FILE**コマンドを呼び出すことを忘れないでください。また、アプリケーションを終了する (または他のデータベースを開く) 場合、4Dは開かれたリソースファイルを自動的に閉じる点にも注目してください。

GET ICON RESOURCE

GET ICON RESOURCE (resID ; resData {; fileRef})

引数	型		説明
resID	倍長整数	⇒	アイコンリソースID番号
resData	ピクチャー	⇒	ピクチャを受け取るピクチャフィールドまたは変数
		⇐	cicnリソースの内容
fileRef	DocRef	⇒	リソースファイル参照番号、または 省略時、開かれている全てのリソースファイル

説明

GET ICON RESOURCE コマンドは、resIDに渡されるIDを持つカラーアイコン ("cicn") リソースに格納されているアイコンをresDataのピクチャフィールドまたはピクチャ変数に返します。

リソースが見つからなかった場合、resDataは変更されず、システム変数OKに0が設定されます。

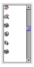
resFileに有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。resFileを省略した場合、リソースチェーン内で最初に見つかったリソースが返されます。

例題

以下の例は、稼働している4Dアプリケーション内に登録されているカラーアイコンをピクチャ配列にロードします:

```
If(On Windows)
   $vh4DResFile:=Open resource file(Replace string(Application file;".EXE";".RSR"))
Else
   $vh4DResFile:=Open resource file(Application file)
End if
RESOURCE LIST("cicn";$alResID;$asResName;$vh4DResFile)
$vlNbIcons:=Size of array($alResID)
ARRAY PICTURE(ag4DIcon;$vlNbIcons)
For($vlElem;1;$vlNbIcons)
   GET ICON RESOURCE($alResID{$vlElem};ag4DIcon{$vlElem};$vh4DResFile)
End for
```

このコード実行後、配列はフォーム上で以下のように表示されます:



システム変数およびセット

リソースが見つかるとOKは1に設定されます。そうでなければ0に設定されます。

Get indexed string

Get indexed string (resID ; strID {; resFile}) -> 戻り値

引数	型		説明
resID	倍長整数	→	リソースID番号、または 'group'要素の'id'属性 (XLIFF)
strID	倍長整数	→	ストリング番号、または 'trans-unit'要素の'id'属性 (XLIFF)
resFile	DocRef	→	リソースファイル参照番号、または 省略時: すべてのXLIFFファイル、または 開かれているリソースファイル
戻り値	文字	↩	インデックス付きストリングの値

説明

Get indexed string コマンドは以下の値を返します:

- ストリングリスト ("STR#") リソースに格納された、IDがresIDの文字列、または
- 開かれたXLIFFファイル中、'group'要素の'id'属性がresIDである文字列 (後述の"XLIFFアーキテクチャとの互換性"を参照)。

strIDにストリングの番号を渡します。ストリングリストリソースのストリングは1からNの順に番号が振られます。ストリングリストリソースのすべてのストリング (およびそのストリング番号) を取得するにはSTRING LIST TO ARRAYを使用します。

リソースまたはそのリソース内のストリングが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

resFileに有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。resFileを省略した場合は、リソースチェーン内で最初に見つかったリソースのオカレンスが返されます。

Note: ストリングリストリソースの各ストリングは、最大255文字を格納できます。

XLIFFアーキテクチャとの互換性

4D v11より、Get indexed string コマンドはXLIFFアーキテクチャと互換があります。コマンドはまずresIDとstrIDに対応するリソースをすべての開かれたXLIFFファイル内で探します (resFile 引数が省略されていれば)。この場合、resIDは**group要素のid属性を表し、**strIDは**trans-unit要素のid属性を表します。**値が見つからない場合、コマンドは引き続き開かれたリソースファイルを検索します。4DにおけるXLIFFアーキテクチャに関する詳細は4D Design Referenceマニュアルを参照してください。

例題

SET MENU BARコマンドの例題参照

システム変数およびセット

リソースが見つかるとOKは1に、そうでなければ0にう設定されます。

GET PICTURE RESOURCE

GET PICTURE RESOURCE (resID ; resData {; resFile})

引数	型	説明
resID	倍長整数	→ リソースID番号
resData	フィールド, 変数	→ ピクチャを受け取る、ピクチャフィールドまたは変数 ← PICTリソースの内容
resFile	DocRef	→ リソースファイル参照番号、または 省略時、開かれている全てのリソースファイル

説明

GET PICTURE RESOURCE コマンドは、resIDのIDを持つピクチャ ("PICT") リソースに格納されているピクチャをresDataのピクチャフィールドまたは変数に返します。

リソースが見つからなかった場合、resDataは変わらず、システム変数OKに0が設定されます。

resFileに有効なリソースファイル参照番号を渡すと、リソースはそのリソースファイル内でのみ検索されます。resFileを省略した場合は、リソースチェーン内で最初に見つかったリソースのオカレンスが返されます。

Note: ピクチャリソースは、少なくとも数メガバイトのサイズになる可能性があります。

例題

RESOURCE LISTコマンドの例題参照

システム変数およびセット

リソースが見つかるとOKは1に、そうでなければ0に設定されます。

エラー管理

ピクチャをロードするための十分なメモリがない場合、エラーが生成されます。ON ERR CALLを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

GET RESOURCE

GET RESOURCE (resType ; resID ; resData {; resFile})

引数	型	説明
resType	文字	→ 4文字のリソースタイプ
resID	倍長整数	→ リソースID番号
resData	BLOB	→ データを受け取るBLOBフィールドまたは変数
		← リソースの内容
resFile	DocRef	→ リソースファイル参照番号、または 省略時、開かれている全てのリソースファイル

説明

GET RESOURCE コマンドは、resTypeとresIDに渡されるタイプとIDを持つリソースの内容を、BLOBフィールドまたはBLOB変数のresDataに返します。

重要: resTypeには4文字を渡す必要があります。.

リソースが見つからなかった場合、resDataはそのまま変わらず、システム変数OKに0が設定されます。

resFileに有効なリソースファイル参照番号を渡すと、リソースはそのリソースファイル内でのみ検索されます。resFileを省略した場合、リソースチェーン内で最初に見つかったリソースのオカレンスが返されます。

Note: リソースは、数メガバイトのサイズになる可能性があります。

プラットフォーム独立性

Mac OSベースのリソースに対して作業していることを忘れないでください。プラットフォームが何であろうと、倍長整数のような内部リソースデータはMacintoshバイトオーダーで格納されます。Windows上では、(ストリングリストリソースおよびピクチャリソース等の) 標準リソースデータは必要に応じて自動的にバイトスワップされます。これに対して、カスタム内部データストラクチャを作成および使用する場合は、BLOB から取り出すデータのバイトスワップは開発者に任されています (BLOB to longintコマンドにMacintosh byte ordering定数を渡すなど)。

例題

SET RESOURCEコマンドの例題参照

システム変数およびセット

リソースが見つかるとOKに1が、そうでなければ0が設定されます。

エラー管理

リソースをロードするための十分なメモリがない場合、エラーが生成されます。ON ERR CALLを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

Get resource name

Get resource name (resType ; resID {; resFile}) -> 戻り値

引数	型		説明
resType	文字	→	4文字のリソースタイプ
resID	倍長整数	→	リソースID番号
resFile	DocRef	→	リソースファイル参照番号、または 省略時、開かれている全てのリソースファイル
戻り値	文字	←	リソースの名前

説明

Get resource nameコマンドは、resTypeのタイプでresIDのIDを持つリソースの名前を返します。

resFileに有効なリソースファイル参照番号を渡すと、リソースはそのリソースファイル内でのみ検索されます。resFileを省略した場合は、現在開かれているすべてのリソースファイル内で検索されます。

リソースが存在しない場合、**Get resource name**は空の文字列を返します。

Get resource properties

Get resource properties (resType ; resID {; resFile}) -> 戻り値

引数	型		説明
resType	文字	→	4文字のリソースタイプ
resID	倍長整数	→	リソースID番号
resFile	DocRef	→	リソースファイル参照番号、または 省略時、開かれている全てのリソースファイル
戻り値	倍長整数	←	リソースの属性

説明

Get resource propertiesコマンドは、resTypeに渡されるタイプかつresIDに渡されるIDを持つリソースの属性を返します。

resFileに有効なリソースファイル参照番号を渡すと、リソースはそのリソースファイル内でのみ検索されます。resFileを省略した場合は、現在開かれているすべてのリソースファイル内で検索されます。

リソースが存在しない場合、**Get resource properties**は0を返し、システム変数OKに0が設定されます。

Get resource propertiesによって返される数値は、ビットが特別の意味を持っているビットフィールド値として理解する必要があります。

例題

Get resource nameの例題参照

システム変数およびセット

リソースが存在しない場合OK変数は0に、そうでなければ1に設定されます。

Get string resource

Get string resource (resID {; resFile}) -> 戻り値

引数	型		説明
resID	倍長整数	→	リソースID番号
resFile	DocRef	→	リソースファイル参照番号、または 省略時、開かれている全てのリソースファイル
戻り値	文字	↩	'STR 'リソースの内容

説明

Get string resource コマンドは、resIDに渡されるIDを持つストリング ("STR ") リソースに格納されている文字列を返します。

リソースが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

resFileに有効なリソースファイル参照番号を渡すと、リソースはそのリソースファイル内でのみ検索されます。resFileを省略した場合は、リソースチェーン内で最初に見つかったリソースの出現が返されます。

Note: ストリングリソースは、最大255バイトの文字を含めることができます。

例題

以下の例は、ストリングリソースID=20911の内容を表示します。このリソースは現在開かれているリソースファイルの少なくとも1つに配置されている必要があります:

```
ALERT(Get string resource(20911))
```

システム変数およびセット

リソースが見つかればOK変数は1に、そうでなければ0に設定されます。

Get text resource

Get text resource (resID {; resFile}) -> 戻り値

引数	型		説明
resID	倍長整数	→	リソースID番号
resFile	DocRef	→	リソースファイル参照番号、または 省略時、開かれている全てのリソースファイル
戻り値	テキスト	↩	TEXTリソースの内容

説明

Get text resource コマンドは、resIDに渡されるIDを持つテキスト ("TEXT") リソースに格納されているテキストを返します。

リソースが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

resFileに有効なリソースファイル参照番号を渡すと、リソースはそのリソースファイル内でのみ検索されます。resFileを省略した場合は、リソースチェーン内で最初に見つかったリソースの出現が返されます。

Note: テキストリソースは、最大32000文字を使用することができます。

例題

以下の例は、テキストリソースID=20800の内容を表示します。このIDは、現在開かれているリソースファイルの少なくとも1つに登録されている必要があります:

```
ALERT(Get text resource(20800))
```

システム変数およびセット

リソースが見つかるとOK変数は1に、そうでなければ0に設定されます。

Open resource file

Open resource file (resFilename {; fileType}) -> 戻り値

引数	型		説明
resFilename	文字	→	リソースファイルのファイル名またはフルパス名, または 空の文字列を指定するとファイルを開くダイアログボックスを表示
fileType	文字	→	Mac OSファイルタイプ (4文字)、または Windowsファイル拡張子(1から3文字)、または 省略時、すべてのファイル
戻り値	DocRef	←	リソースファイル参照番号

説明

Open resource fileコマンドは、resFileNameに渡した名前またはパス名を持つリソースファイルを開きます。

ファイル名を渡す場合、そのファイルはデータベースのストラクチャファイルと同階層に配置されていなければなりません。他のフォルダ内に配置されているリソースを開くには、フルパス名を渡します。

resFileNameに空の文字列を渡すと、ファイルを開くダイアログボックスが表示されます。このダイアログボックスでリソースファイルを選択し、開くことができます。ダイアログボックスを取り消すと、リソースファイルは開かれません。この場合、**Open resource file**はヌルのDocRefを返し、システム変数OKに0を設定します。

デフォルトで、コマンドは引数に渡されたファイルのリソースフォークを開きます。フォークが空の場合、コマンドはそのファイルのデータフォークを開き、そこで見つかるリソースにアクセスします。詳細はリソースを参照してください。

リソースファイルが正常に開かれると、**Open resource file**はそのリソースファイル参照番号を返し、システム変数OKに1を設定します。リソースファイルが存在しない場合や開こうとしているファイルがリソースファイルではない場合、エラーが生成されます。

- Macintosh上でファイルを開くダイアログボックスを使用する場合、デフォルトですべてのファイルが表示されます。特定タイプのファイルを表示するには、オプション引数fileTypeにそのファイルタイプを指定します。
- Windows上でファイルを開くダイアログボックスを使用する場合、デフォルトですべてのファイルが表示されます。ファイルの特定タイプを表示するには、1から3文字のWindowsファイル拡張子またはMAP FILE TYPESコマンドを使ってマップされた任意のMacintoshファイルタイプをオプション引数fileTypeに渡します。

リソースファイルを使い終わったらCLOSE RESOURCE FILEコマンドの呼び出しを忘れないでください。ただし、アプリケーションを終了する (または他のデータベースを開く) 場合は、**Open resource file**を使って開かれたすべてのリソースファイルを4Dが自動的に閉じます。

デフォルトで排他的に読み書きモードでファイルを開くOpen documentコマンドと違って、**Open resource file**は4Dセッション内で既に開かれているリソースファイルを開くことを妨げません。例えばOpen documentを使って同じドキュメントを2度開こうとすると、2度目を開く時にI/Oエラーが返されます。これに対して、4Dセッション内で既に開かれているリソースファイルを開こうとすると、**Open resource file**は既に開いているファイルのリソースファイル参照番号を返します。何度かリソースファイルを開いても、そのリソースファイルを閉じるには1回だけCLOSE RESOURCE FILEを呼び出すだけで済みます。このことは、4Dセッション内でリソースファイルが開かれる場合にのみ有効である点に注意してください。他のアプリケーションで既に開かれているリソースファイルを開こうとすると、I/Oエラーが発生します。

警告:

- 4Dアプリケーションや、4D Desktopをマージしたアプリケーションのリソースファイルへのアクセスは禁止されています。
- 技術的には可能ですが、コンパイルされたり4D Desktopをマージしたデータベースではプログラムコードは動作しなくなるので、データベースストラクチャリソースファイルは使用しないことをお勧めします。
  しかしながら、プログラムをしようしてこのリソースにアクセスし、追加、削除、変更をするつもりなら、ご使用になる動作環境で必ずテストしてください。4D Server上では重大な問題に通じるでしょう。例えばサーバマシン上で (データベースメソッドかストアドプロシージャを使用して) リソースを変更すると、透過的にワークステーションにリソースを分配する4D Serverの管理サービスに確実に影響するでしょう。4D Clientではストラクチャファイルに直接アクセスする手段を持っていないことに注意してください。ストラクチャファイルは、サーバマシンにあるからで す。
- この理由により、リソースを使用する場合、そのリソースは自身のファイルに格納してください。
- 独自のリソースを使って作業する場合、負数のリソースIDを使用してはいけません。これはオペレーションシステムで使用されるために予約されています。また0から14999の範囲のリソースIDを使用してもいけません。この範囲は、4Dで使用されるために予約されている番号です。カスタムリソース には15,000から32,767の範囲の番号を使用してください。あるリソースファイルを開くと、そのファイルがリソースチェーン内で検索される最初の ファイルとなることを覚えておいてください。システムまたは4Dリソースの範囲内のIDを持つリソースをそのファイルに格納すると、GET RESOURCEコマンドや4Dの内部ルーチンはそのリソースを見つけます。これが目的とする結果かもしれませんが、もしそうでない場合は、システムエラーを引き起こすかもしれないため、この範囲の値を使用してはいけません。
- リソースファイルは高度に構造化されたファイルで、1ファイル当たり2,700を超えるリソースを許可しません。数多くのリソースを含んでいるリソースファイルで作業する場合、そのリソースファイルに新しいリソースを追加する前にそのリソースの数を調べるべきです。これについては、RESOURCE TYPE LISTの例題のCount resourcesを参考にしてください。

リソースファイルを開くと、RESOURCE TYPE LISTとRESOURCE LISTを使って、そのファイルの内容を解析できます。

例題 1

Windows上で、以下の例はデータベースフォルダに配置された"MyPrefs.res"リソースファイルを開きます:

```
$vhResFile:=Open resource file("MyPrefs";"res ")
```

Macintosh上で、この例は"MyPrefs"ファイルを開きます。

例題 2

Windows上で、以下の例はデータベースフォルダに配置された"MyPrefs.rsr"リソースファイルを開きます:

```
$vhResFile:=Open resource file("MyPrefs";"rsr")
```

Macintosh上で、この例は"MyPrefs"ファイルを開きます。

例題 3

以下の例は、すべてのファイルタイプを表示するファイルを開くダイアログボックスを表示します:

```
$vhResFile:=Open resource file("")
```

例題 4

以下の例は、デフォルトファイルタイプを使ってCreate resource fileコマンドで作成されたファイルを表示するファイルを開くダイアログボックスを表示します:

```
$vhResFile:=Open resource file("";"res ")
If(OK=1)
   ALERT("You just opened "+Document+".")
   CLOSE RESOURCE FILE($vhResFile)
End if
```

システム変数およびセット

リソースファイルが正しく開かれるとOK変数は1に設定されます。リソースファイルが開けないか、ファイルを開くダイアログボックスがユーザによりキャンセルされると、OKは0に設定されます。

ファイルを開くダイアログを使用してリソースファイルが開かれると、Document変数にはそのファイルへのパスが格納されます。

エラー管理

リソースファイルまたはI/Oの問題でリソースを開くことができなかった場合、エラーが生成されます。ON ERR CALLコマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

RESOURCE LIST

RESOURCE LIST (resType ; resIDs ; resNames {; resFile})

引数	型		説明
resType	文字	→	4文字のリソースタイプ
resIDs	倍長整数配列	←	リソースID番号
resNames	文字配列	←	リソース名
resFile	DocRef	→	リソースファイル参照番号、または 省略時、開かれている全てのリソースファイル

説明

RESOURCE LIST コマンドは、resTypeに渡したタイプのリソースIDとリソース名から、resIDsとresNamesの配列を作成します。

重要: resTypeには4文字を受渡す必要があります。.

オプション引数resFileに有効なリソースファイル参照番号を渡すと、そのファイルのリソースのみが返されます。resFileを省略した場合、現在開いているリソースファイルのすべてのリソースがリストされます。

RESOURCE LISTを呼び出す前に配列を事前定義する場合は、resIDsを倍長整数配列に、resNamesを文字列またはテキスト配列として定義します。配列の事前定義を行わない場合、このコマンドはresIDsを倍長整数配列、resNamesをテキスト配列として作成します。

このコマンドの実行後、Size of arrayをresIDsまたはresNamesの配列に対して適用し、見つかったリソースの数を調べることができます。

例題 1

以下の例は、配列$alResIDと$atResNameにデータベースのストラクチャファイル内に存在するストリングリストリソースのIDと名前を返します:

```
If(On Windows)
   $vhStructureResFile:=Open resource file(Replace string(Structure file;".4DB";".RSR"))
Else
   $vhStructureResFile:=Open resource file(Structure file)
End if
If(OK=1)
   RESOURCE LIST("STR#";$alResID;$atResName;$vhStructureResFile)
End if
```

例題 2

以下の例は、現在開かれているリソースファイル内にあるピクチャリソースをデータベースのピクチャライブラリの中にコピーします:

```
RESOURCE LIST("PICT";$alResID;$atResName)
Open window(50;50;550;120;5;"Copying PICT resources...")
For($vlElem;1;Size of array($alResID))
   GET PICTURE RESOURCE($alResID{$vlElem};$vgPicture)
   If(OK=1)
      $vsName:=$atResName{$vlElem}
      If($vsName="")
         $vsName:="PICT resID="+String($alResID{$vlElem})
      End if
      ERASE WINDOW
      GOTO XY(2;1)
      MESSAGE("Adding picture ""+$vsName+"" to the DB picture library.")
      SET PICTURE TO LIBRARY($vgPicture;$alResID{$vlElem};$vsName)
   End if
End for
CLOSE WINDOW
```

RESOURCE TYPE LIST

RESOURCE TYPE LIST (resTypes {; resFile})

引数	型	説明
resTypes	文字配列	← 利用可能なリストタイプのリスト
resFile	DocRef	→ リソースファイル参照番号、または 省略時、開かれている全てのリソースファイル

説明

RESOURCE TYPE LIST コマンドは、現在開いているリソースファイルの中に存在するリソースのリソースタイプによるresTypes配列を作成します。

オプション引数resFileに有効なリソースファイル参照番号を渡すと、そのファイルのリソースのみが返されます。resFileを省略した場合、現在開いているリソースファイルのすべてのリソースが返されます。

RESOURCE TYPE LISTを呼び出す前に、resTypes配列を文字列配列またはテキスト配列としてあらかじめ定義することができます。配列の事前定義を行わない場合、このコマンドはテキスト配列のresTypesを作成します。

このコマンドの実行後、Size of arrayをresTypes配列に対して実行し、見つかったリソースタイプの数を調べることができます。

例題 1

以下の例は、現在開いているすべてのリソースファイルに存在するリソースのリソースタイプによるatResType配列を作成します:

```
RESOURCE TYPE LIST(atResType)
```

例題 2

以下の例題では、Macintoshの4Dストラクチャファイルに古い4Dプラグインが含まれているかどうかをテストします。もし含まれていれば、これをWindowsで使用するためには更新が必要です。

```
$vhResFile:=Open resource file(Structure file)
RESOURCE TYPE LIST(atResType;$vhResFile)
If(Find in array(atResType;"4DEX")>0)
   ALERT("This database contains old model Mac OS 4D plug-ins."+(Char(13)*2)+
   "You will have to update them for using this database on Windows.")
End if
```

Note: 古いプラグインはストラクチャファイル以外にも存在する可能性があります。またデータベースにはProc.Extファイルが含まれているかもしれません。

例題 3

以下のプロジェクトメソッドは、リソースファイルの中に存在するリソースの数を返します:

```
  ` Count resources プロジェクトメソッド
  ` Count resources ( 時間 ) -> 倍長整数
  ` Count resources ( DocRef ) -> リソース数

C_LONGINT($0)
C_TIME($1)

$0:=0
RESOURCE TYPE LIST($atResType;$1)
For($vlElem;1;Size of array($atResType))
   RESOURCE LIST($atResType{$vlElem};$alResID;$atResName;$1)
   $0:=$0+Size of array($alResID)
End for
```

このメソッドをデータベースに組み込むと、以下のように記述できます:

```
$vhResFile:=Open resource file("")
If(OK=1)
   ALERT("The file “"+Document+"” contains "+String(Count resources($vhResFile))+" resource(s).")
   CLOSE RESOURCE FILE($vhResFile)
End if
```

STRING LIST TO ARRAY

STRING LIST TO ARRAY (resID ; strings {; resFile})

引数	型		説明
resID	倍長整数	⇒	リソースID番号、または 'group'要素の'id'属性 (XLIFF)
strings	文字配列	⇐	STR#リソースから取り出した文字列、または 'group'要素から取り出した文字列 (XLIFF)
resFile	DocRef	⇒	リソースファイル参照番号、または 省略時、開かれているすべてのXLIFFファイル リソースファイル

説明

STRING LIST TO ARRAYコマンドは以下の要素から構築されるstrings配列を生成します:

- ストリングリスト ("STR#") リソースに格納された、IDがresIDの文字列、または
- 開かれたXLIFFファイル中、'group'要素の'id'属性がresIDである文字列 (後述の"XLIFFアーキテクチャとの互換性"を参照)。

リソースが見つからない場合strings配列はそのまま変更されず、システム変数OKに0が設定されます。

resFileに有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。resFileを省略した場合は、リソースチェーン内で最初に見つかったリソースのオカレンスが返されます。

STRING LIST TO ARRAYを呼び出す前に、配列stringsを文字列またはテキスト配列として宣言できます。配列を事前に定義しない場合、テキスト配列として作成されます。

注: ストリングリストリソースの各ストリングは、最大255文字を格納できます。

Tip: ストリングリストの総サイズを32Kに、また1リソースあたり数百文字列に制限しなければなりません。

XLIFFアーキテクチャとの互換性

4D v11より、**STRING LIST TO ARRAY**コマンドはXLIFFアーキテクチャと互換があります。コマンドはまずresIDとstrIDに対応するリソースをすべての開かれたXLIFFファイル内で探します (resFile 引数が省略されていれば)。この場合、resIDは**group**要素の**id**属性を表し、strIDは**trans-unit**要素の**id**属性を表します。値が見つからない場合、コマンドは引き続き開かれたリソースファイルを検索します。4DにおけるXLIFFアーキテクチャに関する詳細は4D Design Referenceマニュアルを参照してください。

システム変数およびセット

リソースが見つかるとOK変数に1が、そうでなければ0が設定されます。

ARRAY TO STRING LIST

ARRAY TO STRING LIST (strings ; resID {; resFile})

引数	型		説明
strings	文字配列	→	文字列またはテキスト配列 (新しいSTR#リソースの内容)
resID	倍長整数	→	リソースID番号
resFile	DocRef	→	リソースファイル参照番号, または 省略時、現在のリソースファイル

説明

互換性に関する注意: リソースへの書き込みを行うコマンドは動作しません。

Create resource file

Create resource file (resFilename {; fileType {; *}}) -> 戻り値

resFilename	文字	→	リソースファイルのファイル名またはフルパス名、または 空の文字列を指定するとファイル保存ダイアログボックスを表示
fileType	文字	→	Mac OSファイルタイプ (4文字)、または Windowsファイル拡張子(1から3文字)、または 省略時、リソースドキュメント("res " / .RES)
*		→	渡した場合、データフォーク

説明

互換性に関する注意: リソースへの書き込みを行うコマンドは動作しません。

DELETE RESOURCE

DELETE RESOURCE (resType ; resID {; resFile})

引数	型		説明
resType	文字	→	4文字のリソースタイプ
resID	倍長整数	→	リソースID番号
resFile	DocRef	→	リソースファイル参照番号、または 省略時、カレントリソースファイル

説明

互換性に関する注意: リソースへの書き込みを行うコマンドは動作しません。

Get component resource ID

Get component resource ID (compName ; resType ; originalResNum) -> 戻り値

引数	型		説明
compName	文字	⇒	リソースを参照するコンポーネント名
resType	文字	⇒	リソースタイプ(4文字), PICT または STR#
originalResNum	倍長整数	⇒	コンポーネントとしてインストールされる前の オリジナルのリソース番号
戻り値	倍長整数	⇐	カレントリソース番号

説明

互換性メモ: このコマンドは4D v11以降とは互換のない前世代のコンポーネントで使用します。現在では効果はなく、使用されるべきではありません。

SET PICTURE RESOURCE

SET PICTURE RESOURCE (resID ; resData {; resFile})

引数	型		説明
resID	倍長整数	⇒	リソースID番号
resData	ピクチャー	⇒	新しいPICTリソースの内容
resFile	DocRef	⇒	リソースファイル参照番号、または 省略時、カレントリソースファイル

説明

互換性に関する注意: リソースへの書き込みを行うコマンドは動作しません。

SET RESOURCE

SET RESOURCE (resType ; resID ; resData {; resFile})

引数	型		説明
resType	文字	⇒	4文字のリソースタイプ
resID	倍長整数	⇒	リソースID番号
resData	BLOB	⇒	リソースの新しい内容
resFile	DocRef	⇒	リソースファイル参照番号、または 省略時、カレントリソースファイル

説明

互換性に関する注意: リソースへの書き込みを行うコマンドは動作しません。

SET RESOURCE NAME

SET RESOURCE NAME (resType ; resID ; resName {; resFile})

引数	型		説明
resType	文字	→	4文字のリソースタイプ
resID	倍長整数	→	リソースID番号
resName	文字	→	新しいリソース名
resFile	DocRef	→	リソースファイル参照番号、または 省略時、カレントリソースファイル

説明

互換性に関する注意: リソースへの書き込みを行うコマンドは動作しません。

SET RESOURCE PROPERTIES

SET RESOURCE PROPERTIES (resType ; resID ; resAttr {; resFile})

引数	型		説明
resType	文字	→	4文字のリソースタイプ
resID	倍長整数	→	リソースID番号
resAttr	倍長整数	→	リソースの新しい属性
resFile	DocRef	→	リソースファイル参照番号、または 省略時、カレントリソースファイル

説明

互換性に関する注意: リソースへの書き込みを行うコマンドは動作しません。

SET STRING RESOURCE

SET STRING RESOURCE (resID ; resData {; resFile})

引数	型		説明
resID	倍長整数	⇒	リソースID番号
resData	文字	⇒	新しいSTR リソースの内容
resFile	DocRef	⇒	リソースファイル参照番号、または 省略時、カレントリソースファイル

説明

互換性に関する注意: リソースへの書き込みを行うコマンドは動作しません。

SET TEXT RESOURCE

SET TEXT RESOURCE (resID ; resData {; resFile})

引数	型		説明
resID	倍長整数	⇒	リソースID番号
resData	文字	⇒	TEXTリソースの新しい内容
resFile	DocRef	⇒	リソースファイル参照番号、または 省略時、カレントリソースファイル

説明

互換性に関する注意: リソースへの書き込みを行うコマンドは動作しません。

リレーション

- リレーションについて
- CREATE RELATED ONE
- GET AUTOMATIC RELATIONS
- GET FIELD RELATION
- OLD RELATED MANY
- OLD RELATED ONE
- RELATE MANY
- RELATE MANY SELECTION
- RELATE ONE
- RELATE ONE SELECTION
- SAVE RELATED ONE
- SET AUTOMATIC RELATIONS
- SET FIELD RELATION

リレーションについて

この章のコマンド、特にRELATE ONEとRELATE MANYは自動リレートおよび自動でないリレートを設定、管理するものです。この章のコマンドを使用する前に、テーブル間のリレートの作成について4D Design Referenceマニュアルを参照してください。

テーブルの自動リレートを利用するコマンド

2つのテーブルは自動リレートで関連付けることができます。一般的にテーブルの自動リレートは、リレート先テーブルのレコードをロード、または選択するために使用します。リレートを使用することで多くの処理を実行することができます。

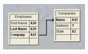
以下のような処理が含まれます:

- データ入力
- 出力フォームによる画面上のリスト出力
- 帳票の印刷
- セレクションに対する検索、ソート、フォーミュラでの更新

性能を最大限に引き出すために、4Dは1レコードがテーブルのカレントレコードになる場合にのみ自動リレーションを実行します。上記のリストに示した各操作を行ったときに、以下の原則に従って、リレートレコードがロードされます:

- リレーションがリレートしたテーブルの1つのレコードのみを選択する場合、そのレコードはディスクからロードされます。
- リレーションがリレートしたテーブルの複数のレコードを選択する場合、新しいレコードセレクションがそのテーブルに作成され、セレクション内の先頭レコードがディスクからロードされます。

以下の図のテーブルストラクチャを例にとってみましょう。[Employees]テーブルをデータ入力のためにロードした場合は、[Companies]テーブルからリレート先レコードが選択され、ロードされます。同様に[Companies]テーブルをロードしてデータ入力用に表示した場合には、[Employees]テーブルから関連するレコードが選択されます。



上図において、[Employees]テーブルは**nテーブル**として参照され、[Companies]テーブルは**1テーブル**として参照されます。これを解かりやすくするために、多くの人が1つの会社に関連しており、各会社には多くの人がいると考えてください。

同様に[Employees]Companyフィールドは**nフィールド**として参照され、[Companies]Nameフィールドは**1フィールド**として参照されます。

リレートフィールドが、いつもユニーク(重複しない)とは限りません。例えば、[Companies]Nameフィールドは同じ値を含んだ会社のレコードをいくつも持っているかもしれません。こういう場合は、常にユニークであるリレート先テーブルの別のフィールドでリレート作成することにより、簡単に処理することができます。このようなフィールドの例として、会社IDフィールドがあります。

以下の表に掲げたコマンドは、コマンドの実行中にリレート先のリレートレコードをロードするために自動リレートを使用します。これらのコマンドはすべてn対1の自動リレートを使用します。1対nの欄が○になっているコマンドのみが自動の1対nリレートに対応します。

コマンド	1対n
ADD RECORD	O
ADD SUBRECORD	X
APPLY TO SELECTION	X
DISPLAY SELECTION	X
EXPORT DIF	X
EXPORT SYLK	X
EXPORT TEXT	X
EXPORT DATA	X
MODIFY RECORD	O
MODIFY SUBRECORD	X
MODIFY SELECTION	O (データ入力中)
ORDER BY	X
ORDER BY FORMULA	X
QUERY BY FORMULA	O
QUERY SELECTION	O
QUERY	O
PRINT LABEL	X
PRINT SELECTION	O
QR REPORT	X
SELECTION TO ARRAY	X
SELECTION RANGE TO ARRAY	X

テーブルリレートを実行するコマンド

自動リレートとは、コマンドがレコードをロードするたびに、そのテーブルに関連する1つまたは複数のレコードを自動的に選択するという意味ではありません。レコードをロードするコマンドを使用した後で、リレート先データにアクセスする必要がある場合、RELATE ONEやRELATE MANYを使用して、リレート先レコードを明示的に選択することが必要なケースもあります。

前ページの表に掲げたコマンドの一部(QUERYコマンド等)は、処理の終了後にカレントレコードをロードします。この場合、ロードされたレコードは、リレート先の関連するレコードを自動的には選択しません。ここでも、リレート先データにアクセスする必要がある場合、RELATE ONEやRELATE MANYを使用してリレート先レコードを明示的に選択する必要があります。

CREATE RELATED ONE

CREATE RELATED ONE (aField)

引数	型		説明
aField	フィールド	→	nフィールド

説明

CREATE RELATED ONEには2つの機能があります。関連するレコードがaField に対して存在しない場合 (つまりaField の現在の値に一致するものがない場合) 、CREATE RELATED ONEは新しくリレート先レコードを作成します。

適切なフィールドに値を保存するには、nフィールドから1フィールドへ値を割り当てます。SAVE RELATED ONEを実行して、この新しいレコードを保存します。

リレート先レコードが存在する場合、CREATE RELATED ONEはRELATE ONEコマンドと全く同じようにそのレコードをメモリにロードします。

GET AUTOMATIC RELATIONS

GET AUTOMATIC RELATIONS (one ; many)

引数	型		説明
one	ブール	⇐	すべてのn対1リレートの状態
many	ブール	⇐	すべての1対nリレートの状態

説明

GET AUTOMATIC RELATIONS コマンドにより、データベースのすべてのマニュアルn対1リレートおよび1対nリレートに関する自動/マニュアルのステータスがカレントプロセスにおいて変更されたかどうかを知ることができます。

- one: 前回のSET AUTOMATIC RELATIONSコマンドの呼び出しにより、すべてのマニュアルn対1リレートが自動に設定された場合、この引数はTrueを返します - 例えばSET AUTOMATIC RELATIONS(True;False)。
  SET AUTOMATIC RELATIONSコマンドが呼び出されなかった場合や、前回の実行によりマニュアルn対1リレートが変更されなかった場合にはFalseを返します - 例えばSET AUTOMATIC RELATIONS(False;False)。
- many: 前回のSET AUTOMATIC RELATIONSコマンドの呼び出しにより、すべてのマニュアル1対nリレートが自動に設定された場合、この引数はTrueを返します - 例えばSET AUTOMATIC RELATIONS(True;True)。
  SET AUTOMATIC RELATIONSコマンドが呼び出されなかった場合や、前回の実行によりマニュアル1対nリレートが変更されなかった場合にはFalseを返します - SET AUTOMATIC RELATIONS(True;False)。

例題

GET FIELD RELATION コマンドの例題を参照

GET FIELD RELATION

GET FIELD RELATION (manyField ; one ; many {; *})

引数	型		説明
manyField	フィールド	⇨	リレート開始フィールド
one	倍長整数	⇦	n対1リレートの状態
many	倍長整数	⇦	1対nリレートの状態
*	演算子	⇨	指定時: oneとmanyにはリレーション の現在の状態が返る (値は2または3のみ) 省略時 (デフォルト): プログラムでリレーションが変更されていなければ、oneとmanyに1が返される

説明

GET FIELD RELATIONを使用すると、カレントプロセスの、manyFieldから開始するリレートの自動/マニュアルのステータスを調べることができます。ストラクチャウィンドウで設定した自動リレートをはじめとして、あらゆるリレートを調べることができます。

- manyFieldには、状態を調べようとするリレートが開始するnテーブルのフィールド名を渡します。フィールドmanyFieldから開始するリレートが存在しない場合、引数oneとmanyには0が返されます。またエラーが返されて、システム変数OKには0が代入されます (後述)。
- コマンドの実行後、引数oneには、指定したn対1リレートが自動に設定されているかどうかを示す値が格納されます:
  0 = manyFieldから始まるリレートが存在しません。シンタックスエラー16 ("このフィールドにはリレートが設定されていません。") が生成され、システム変数OKには0が代入されます。
  1 = 指定されたn対1リレートの自動/マニュアルのステータスは、デザインモードのリレートプロパティ内の**自動 1対 1リレート**オプションにより設定されたものです (プログラムにより変更されていない)。
  2 = そのプロセスのn対1のリレートはマニュアルです。
  3 = そのプロセスのn対1のリレートは自動です。
- コマンドの実行後、引数manyには、指定した1対nリレートが自動に設定されているかどうかを示す値が格納されます:
  0 = manyFieldから始まるリレートが存在しません。シンタックスエラー16 ("このフィールドにはリレートが設定されていません。") が生成され、システム変数OKには0が代入されます。
  1 = 指定された1対nリレートの自動/マニュアルのステータスは、デザインモードのリレートプロパティ内の**自動 1対 nリレート**オプションにより設定されたものです (プログラムにより変更されていない)。
  2 = そのプロセスの1対nのリレートはマニュアルです。
  3 = そのプロセスの1対nのリレートは自動です。

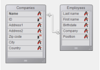
oneおよびmany引数に返された値は、"" テーマの定数と比較することができます:

定数	型	値	コメント
Automatic	倍長整数	3	カレントプロセスに対し、リレートを自動に設定する。
Manual	倍長整数	2	カレントプロセスに対し、リレートをマニュアルに設定する。
No relation	倍長整数	0	
Structure configuration	倍長整数	1	アプリケーションのストラクチャウインドウで指定されたリレートの設定を使用する。

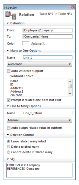
- オプションの引数 * を使用すると、プログラムから修正されていない場合でも、リレートのカレントステータスを"強制的に"読み込むことができます。言い換えれば引数 * を渡した場合、引数oneおよびmanyには値2または3だけが返されます。

例題

以下のようなストラクチャがあります:



[Employees]CompanyフィールドからCompanies]Nameフィールドへリンクするリレートのプロパティは次の通りです:



From: [Employees]Company
To: [Companies]Name
Color: Automatic
Name: Link_1
Automatic
☐ Auto Wildcard support
Wildcard Choice: Name, ID, Address1, Address2, Zip code
☑ Prompt if related one does not exist
Name: Link_1_return
Manual
☐ Auto assign related value in subform
● Leave related many intact
○ Delete related many
○ Cannot delete if related many
FOREIGN KEY: Company
REFERENCES: Company

次のコードはGET FIELD RELATION、GET AUTOMATIC RELATIONS、SET FIELD RELATION、およびSET AUTOMATIC RELATIONSにより提供されるさまざまな機能とともに、その効果を示します:

```
GET AUTOMATIC RELATIONS(one;many) `False, Falseを返す
GET FIELD RELATION([Employees]Company;one;many) `1,1を返す
GET FIELD RELATION([Employees]Company;one;many;*) `3,2を返す

SET FIELD RELATION([Employees]Company;2;0) `n対1リレーションをマニュアルに変更

GET FIELD RELATION([Employees]Company;one;many) `2,1を返す
GET FIELD RELATION([Employees]Company;one;many;*) `2, 2を返す

SET FIELD RELATION([Employees]Company;1;0) `デザインモードで設定された
 `n対1リレーションに戻す

GET FIELD RELATION([Employees]Company;one;many) `1,1を返す
GET FIELD RELATION([Employees]Company;one;many;*) `3,2を返す

SET AUTOMATIC RELATIONS(True;True) `すべてのテーブルのすべてのリレーションを自動に設定

GET AUTOMATIC RELATIONS(one;many) `True, Trueを返す
GET FIELD RELATION([Employees]Company;one;many) `1,1を返す
GET FIELD RELATION([Employees]Company;one;many;*) `3,3を返す
```

OLD RELATED MANY

OLD RELATED MANY (aField)

引数	型		説明
aField	フィールド	→	1フィールド

説明

OLD RELATED MANYは、1フィールドの更新前の内容を使用してリレート処理を実行する以外は、RELATE MANYコマンドと同じ処理を行います。

Note: OLD RELATED MANYコマンドはOld関数により返される、nフィールドの古い値を使用します。詳細は、Old関数の説明を参照してください。

OLD RELATED MANYはリレートテーブルのセレクションを変更し、カレントレコードとしてそのセレクションの最初のレコードを選択します。

OLD RELATED ONE

OLD RELATED ONE (aField)

引数	型	説明
aField	フィールド	→ nフィールド

説明

OLD RELATED ONEは、リレーションを実行する際にaFieldの古い値を使用することを除き、RELATE ONEと同じ処理を行います。

Note: OLD RELATED ONEはOldコマンドにより返される、nフィールドの古い値を使用します。詳細はOldコマンドの説明を参照してください。

OLD RELATED ONEは、更新前のカレントレコードの内容にリレートしたレコードをロードし、そのリレートレコードにアクセスできるようにします。この更新前の関連レコードを修正し、保存したい場合には、SAVE RELATED ONEコマンドを使用する必要があります。新しく作成されたレコードは、更新前の関連するレコードを持たないという点に注意してください。

システム変数およびセット

コマンドが正しく実行されリレーとされたレコードがロードされると、OKシステム変数は1に設定されます。ユーザが(リレートされたレコードが更新されたときに表示される) レコード選択ダイアログボックスで**キャンセル**をクリックすると、OK変数は0に設定されます。

RELATE MANY

RELATE MANY (oneTable | Field)

引数	型		説明
oneTable | Field	テーブル, フィールド	→	すべての1対nリレーションを実行するテーブル または1フィールド

説明

RELATE MANYには、2つの形式があります。

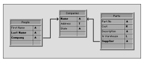
第1の形式RELATE MANY(oneTable)は、oneTableに対してすべての1対nのリレートを実行します。このコマンドは、oneTableに対して1対nのリレートを持つ各テーブルのカレントセレクションを更新します。nテーブルのカレントセレクションは、1テーブルのそれぞれのリレート先フィールドの現在値を反映します。このコマンドが実行される度に、nテーブルのカレントセレクションが再構成され、セレクションの最初のレコードがカレントレコードとしてロードされます。

第2の形式RELATE MANY(oneField)は、oneFieldに対して1対nのリレートを実行します。これは、oneFieldと関連を持つテーブルのみに対しカレントセレクションとカレントレコードを変更します。つまり、リレート先テーブルの関連するレコードがnテーブルのカレントセレクションになることを意味します。

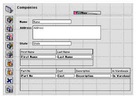
Note: RELATE MANYを実行する際に、1テーブルのカレントセレクションが空の場合、このコマンドは何も行いません。

例題

以下の例は、3つのテーブルが自動リレートで関連付けられています。[People]テーブルと[Parts]テーブルは両方とも[Companies]テーブルに対してn対1のリレート関係にあります。


People
First Name A
Last Name A
Company A
Companies
Name A
Address T
State A
Parts
Part No A
Cost R
Description A
In Warehouse L
Supplier A

[Companies]テーブルのこのフォームは、[People]テーブルと[Parts]テーブル両方の関連するレコードを表示します。


Companies
Name
Address
State
First Name
Last Name
Part No
Cost
Description
In Warehouse

このフォームを表示すると、[People]テーブルと[Parts]テーブルの関連するレコードがロードされ、それぞれのテーブルのカレントセレクションとなります。

他方プログラムで[Companies]テーブルのレコード選択した場合には、リレート先レコードはロードされません。このような場合にはRELATE MANYコマンドを使用する必要があります。

Notes:

- RELATE MANYが空のセレクションに適用されたとき、コマンドは実行されず、nテーブルのセレクションは変わりません。
- コマンドが動作するためには、nフィールドにインデックス属性が付いている必要があります。

以下のメソッドは、[Companies]テーブルの各レコードに対し、警告ウィンドウを表示します。警告ウィンドウには社員数([People]テーブル中の関連するレコードの数)、供給する部品の種類数([Parts]テーブル中の関連するレコードの数)を表示します。この例では、ALERTコマンドの引数が複数行にわたっています。

自動リレートの場合でもRELATE MANYコマンドが必要なことに注目してください。

```
ALL RECORDS([Companies]) ` Select all records in the テーブル
ORDER BY([Companies];[Companies]Name) ` Order records in alphabetical order
For($i;1;Records in table([Companies])) ` Loop once for each record
   RELATE MANY([Companies]Name) ` Select the related records
   ALERT("Company: "+[Companies]Name+Char(13)+"People in company: "
   +String(Records in selection([People]))+Char(13)+
   "数値 of parts they supply: "+String(Records in selection([Parts])))
   NEXT RECORD([Companies]) ` Move to the next record
End for
```

RELATE MANY SELECTION

RELATE MANY SELECTION (aField)

引数	型		説明
aField	フィールド	→	nテーブルのフィールド (リレーションの開始元)

説明

RELATE MANY SELECTION コマンドは、1テーブルのレコードセレクションを元にしてnテーブルのレコードセレクションを作成し、nテーブルの一番目のレコードをカレントレコードとしてロードします。

Note: RELATE MANY SELECTIONは、1テーブルのカレントレコードを変更します。

例題

以下の例では、未収金が$1,000以上の顧客を対象に作成されたすべての請求書を選択しています。[Invoices]テーブルの[Invoices]Customer IDフィールドは、[Customers]テーブルの[Customers]IDフィールドにリレートしています。

```
  ` 顧客を選択
QUERY([Customers];[Customers]Credit>=1000)
  ` これらの顧客に関連するすべての請求書を選択
RELATE MANY SELECTION([Invoices]Customer ID)
```

RELATE ONE

RELATE ONE (manyTable | Field {; choiceField})

引数	型	説明
manyTable | Field	テーブル, フィールド	→ すべての自動リレーションを実行するテーブル, または1テーブルへのマニュアルリレーションが引かれたフィールド
choiceField	フィールド	→ 1テーブルの選択フィールド

説明

RELATE ONEには2つの形式があります。

一番目の形式、RELATE ONE**(manyTable)**は、カレントプロセスのmanyTableに対しすべての自動n対1リレーションを実行します。これはつまり、manyTableの自動n対1リレーションを持つフィールドごとに、コマンドはリレートしたテーブルのリレートしたレコードを選択します。これはプロセスにおいて、リレートしたテーブルのカレントレコードを変更します。

二番目形式RELATE ONE**(manyField{;choiceField})**は、manyFieldに関連するレコードを検索します。自動リレートである必要はありません。レコードが存在する場合、RELATE ONEはリレート先レコードをメモリにロードし、これをそのテーブルのカレントレコードおよびカレントセレクションにします。

オプション引数のchoiceFieldは、manyFieldが文字あるいはテキストフィールドである場合にのみ指定することができます。choiceFieldはリレート先テーブルのフィールドでなければなりません。choiceFieldは文字、テキスト、数値、日付、時間、またはブールフィールドでなければなりません。ピクチャやBLOBを選択することはできません。

choiceFieldが指定され、リレート先テーブルで複数のレコードを発見した場合、RELATE ONEはmanyFieldの内容と一致するレコードを選択リストに表示します。この選択リストは、左の欄にリレート先フィールドの内容を、右の欄にchoiceFieldの内容を表示します。

manyFieldの内容がワイルドカード記号 (@) で終了する場合、複数のレコードが見つかることがあります。一致するレコードが1件しかなければ、リストは表示されません。choiceFieldを指定することは、テーブルのリレートを設定する時点でワイルドカード選択を指定するのと同じことです。ワイルドカード選択に関する詳細は、4D Design Referenceマニュアルを参照してください。

例題

ここに[Invoice] テーブルと[Customers] テーブルが2つのマニュアルリレートにより関連づけられています。ひとつは[Invoice]Bill toから [Customers]IDへ、もうひとつは[Invoice]Ship toから[Customers]IDにリレートされています。

両方のリレート先は同じ[Customers]テーブルになっており、同時に請求先と発送先の情報を得ることは出来ません。フォームに両方の住所を表示するためには、変数とRELATE ONEコマンドを使用します。もし、[Customers]フィールドを表示に使用したなら、一方のリレートから得られたデータしか表示されません。

以下は[Invoice]Bill toと[Invoice]Ship toフィールドのオブジェクトメソッドです。これらはフィールドに入力されると実行されます。

[Invoice]Bill to フィールドのオブジェクトメソッド:

```
RELATE ONE([Invoice]Bill to)
vAddress1:=[Customers]Address
vCity1:=[Customers]City
vState1:=[Customers]State
vZIP1:=[Customers]ZIP
```

[Invoice]Ship to フィールドのオブジェクトメソッド:

```
RELATE ONE([Invoice]Ship to)
vAddress2:=[Customers]Address
vCity2:=[Customers]City
vState2:=[Customers]State
vZIP2:=[Customers]ZIP
```

システム変数およびセット

コマンドが正しく実行されリレートされたレコードがロードされると、OKシステム変数は1に設定されます。ユーザが(リレートされたレコードが更新されたときに表示される) レコード選択ダイアログボックスで**キャンセル**をクリックすると、OK変数は0に設定されます。

RELATE ONE SELECTION

RELATE ONE SELECTION (manyTable ; oneTable)

引数	型	説明
manyTable	テーブル	⇒ nテーブル (リレーションの開始元)
oneTable	テーブル	⇒ 1テーブル (リレーションの参照先)

説明

RELATE ONE SELECTIONコマンドは、manyTableのレコードセレクションをもとにして、oneTableテーブルの新しいセレクションを作成し、その新しいセレクションの最初のレコードをカレントレコードとしてロードします。

このコマンドはmanyTableからoneTableへのリレートがある場合にのみ使用できます。**RELATE ONE SELECTION**はリレートの複数レベルを対象に動作できます。manyTableとoneTableの間には複数のリレートテーブルがある場合があります。これらのリレートは、マニュアルリレートまたは自動リレートのどちらでも動作します。

RELATE ONE SELECTIONは開始テーブルからリレート先テーブルまで設定されているリレーションパスのうち、最も短いものを使用します。同じサイズのパスが存在する場合、**RELATE ONE SELECTION**は開始テーブル中で作成された順番のなかで最初に見つかったパスを使用します。

例題

以下の例では、今日が請求書の支払期日であるすべての顧客を検索します。

以下は、[Invoices]テーブル内のレコードセレクションに基づき、[Customers]テーブルのセレクションを作成します:

```
CREATE EMPTY SET([Customers];"Payment Due")
QUERY([Invoices];[Invoices]DueDate=Current date)
While(Not(End selection([Invoices])))
   RELATE ONE([Invoices]CustID)
   ADD TO SET([Customers];"Payment Due")
   NEXT RECORD([Invoices])
End while
```

以下の手法では**RELATE ONE SELECTION**を使用して同じ結果を得ています:

```
QUERY([Invoices];[Invoices]DueDate=Current date)
RELATE ONE SELECTION([Invoices];[Customers])
```

注: バージョン11以降、このコードはパフォーマンスを低下させることなく以下のコードに書き換えることができます:

```
QUERY([Customers];[Invoices]DueDate=Current date)
```

SAVE RELATED ONE

SAVE RELATED ONE (aField)

引数	型	説明
aField	フィールド	→ nフィールド

説明

SAVE RELATED ONEは、aFieldにリレートするレコードを保存します。CREATE RELATED ONEで新しく作成したレコードを更新する、またはRELATE ONEでロードし修正したレコードを保存するために、SAVE RELATED ONEコマンドを実行します。

SAVE RELATED ONEはサブテーブルに対しては適用できません。親レコードを保存すればサブレコードも自動的に保存されるからです。

SAVE RELATED ONEは、ロックされたレコードを保存しません。このコマンドを使用する場合、最初にレコードがロックされていないことを確認する必要があります。レコードがロックされている場合、このコマンドは無視され、レコードを保存せずエラーも返しません。

SET AUTOMATIC RELATIONS

SET AUTOMATIC RELATIONS (one {; many})

引数	型	説明
one	ブール	⇒すべてのn対1リレーションの状態
many	ブール	⇒すべての1対nリレーションの状態

説明

SET AUTOMATIC RELATIONSは、カレントプロセスで、データベース全体のマニュアルリレートを一時的に自動リレートに変更します。リレートは、次にSET AUTOMATIC RELATIONSコマンドを使用するまで自動リレートのままになります。

- oneがTrueの場合、すべてのn対1のマニュアルリレートを自動リレートに設定します。oneがFalseの場合、前もって自動リレートにしたすべてのn対1リレートがマニュアルリレートに戻ります。
- manyも1対nリレートに対して同じように作用します。

このコマンドはデザインモードで既に自動リレートに設定されたものに対しては無効です。

すべてのリレーションをデザインモードでマニュアルに設定した場合、このコマンドを使用して自動リレートを必要とする処理の直前で自動リレートに切り替えることができます (リレーショナル検索やソート等)。処理が終了した後で、再度マニュアルリレートに戻すことができます。

このコマンドはデザインモードで自動リレートに設定されたリレートを自動リレートを使用した処理(例えばリレートを利用した検索や並び替えなど)を行う直前に手動リレートに変更します。処理が終了した後に、再度**SET AUTOMATIC RELATIONS**を呼び出して手動リレートに戻すことができます。

注: SET AUTOMATIC RELATIONSコマンドに True を渡すと、全ての手動リレートがセッション中に自動モードが"ロック"されます。この場合、SET FIELD RELATION コマンドは**SET AUTOMATIC RELATIONS**の前にあっても後ろにあっても、同じセッション中は呼び出しが無視されます。自動モードを解除してSET FIELD RELATIONの呼び出しを有効化する際には、**SET AUTOMATIC RELATIONS**に Falseを渡します。

例題

以下の例は、すべてのn対1のマニュアルリレートを自動リレートに設定し、前もって自動リレートにした1対nリレートを手動リレートに戻します:

```
SET AUTOMATIC RELATIONS(True;False)
```

SET FIELD RELATION

SET FIELD RELATION (manyTable | manyField ; one ; many)

引数	型	説明
manyTable | manyField	テーブル, フィールド	→ リレーションの開始テーブル, または リレーションの開始フィールド
one	倍長整数	→ テーブルまたはフィールドを開始点とする n対1リレーションの状態
many	倍長整数	→ テーブルまたはフィールドを開始点とする 1対nリレーションの状態

説明

SET FIELD RELATION コマンドを使用すると、デザインモードのリレートプロパティウィンドウで設定した初期状態が何であれ、データベースの各リレートの自動/マニュアルの状態を個別に設定できます。

1番目の引数には、テーブルやフィールドの名前を渡します:

- フィールド名 (manyField) を渡すと、コマンドは指定したnフィールドから開始するリレートに対してのみ適用されます。
- テーブル名 (manyTable) を渡すと、コマンドは指定したnテーブルから開始するリレートに対してのみ適用されます。
- manyFieldまたはmanyTableから開始するリレートが存在しない場合、シンタックスエラー16 (“このフィールドにはリレートが設定されていません。”) が生成され、システム変数OKには0が代入されます。

引数oneとmanyには、指定した1対nリレートやn対1リレートに適用される自動/マニュアルの状態を示す値を渡します。この値として、“”テーマの定数を使用できます。

定数	型	値	コメント
Automatic	倍長整数	3	カレントプロセスに対し、リレートを自動に設定する。
Do not modify	倍長整数	0	リレートの現在のステータスを変更しない。
Manual	倍長整数	2	カレントプロセスに対し、リレートをマニュアルに設定する。
Structure configuration	倍長整数	1	アプリケーションのストラクチャウインドウで指定されたリレートの設定を使用する。

Note: このコマンドを使用して行った変更は、カレントプロセスに対してのみ適用されます。リレートプロパティウィンドウのオプションを用いて指定されたリレート設定は変更されません。

Note: SET AUTOMATIC RELATIONSコマンドに対してを同じセッション中に渡したとき、**SET FIELD RELATION**への呼び出しは、それがSET AUTOMATIC RELATIONSの前にあろうと後ろにあろうと無視されます。自動モードをロックして**SET FIELD RELATION**への呼び出しを有効化するためには、SET AUTOMATIC RELATIONSにを渡します。

例題

このコマンドを使用することにより、クイックレポートのリレート管理がさらに容易になります。以前のバージョンの4Dでは、エディタ上でリレートを利用するには、すべてのリレートを自動に設定する必要がありました。次のコードを使用すると、有用なリレートだけを自動に設定できるようになります:

```
SET AUTOMATIC RELATIONS(False;False)  `すべてのリレーションをリセット
  `以下のリレーションのみを使用する
SET FIELD RELATION([Invoices]Cust_IDt;Automatic;Automatic)
SET FIELD RELATION([Invoice_Row]Invoice_ID;Automatic;Automatic)
QR REPORT([Invoices];Char(1);True;True;True)
```

レコード

- レコード番号について
- レコードスタックの使用
- CREATE RECORD
- DELETE RECORD
- DISPLAY RECORD
- DUPLICATE RECORD
- GOTO RECORD
- Is new record
- Is record loaded
- Modified record
- POP RECORD
- PUSH RECORD
- Record number
- Records in table
- SAVE RECORD
- Sequence number

レコード番号について

レコードに関連した番号には次の3つがあります:

- レコード番号
- レコード位置番号
- シーケンス番号

レコード番号

レコード番号は、レコードに対する絶対/物理的な番号です。この番号はレコードが作成されるごとに自動的に付けられ、レコードが削除されるまで変わることはありません。レコード番号はゼロから始まります。レコード番号は、削除されたレコード番号が新しいレコードに再利用されるため、ユニークではありません。また、レコード番号はデータベースを圧縮、修復すると変更されてしまいます。

レコード位置番号

レコード位置番号は、カレントセレクション中のレコードの位置を示す番号です。したがって、この番号はカレントセレクションに依存します。セレクションを変更またはソートすると、レコード位置番号は変更されます。レコード位置番号は1から始まります。

シーケンス番号

シーケンス番号は、(**自動インクリメント**プロパティ、SQLのAUTO_INCREMENET属性、またはSequence numberコマンドを使用して) レコードのフィールドに割り当てることができる、重複しない番号です。この番号は各レコードに自動で格納されるものではありません。シーケンス番号はデフォルトで1から始まり、新しいレコードを作成するごとに加算されていきます。レコード番号と違って、シーケンス番号はレコードが削除されたり、データベースが圧縮や修復されても再利用されません。シーケンス番号を使用して、レコードに重複しないID番号を持たせることができます。トランザクション中に加算されたシーケンス番号は、トランザクション処理が取り消されても、減少することはありません。

Note: SET DATABASE PARAMETERコマンドを使用してテーブルの内部的なカウンタを変更しても、4Dは全くチェックを行いません。このカウンタを減少させた場合、新しいレコードは既に割り当てられた番号を使用して作成されます。

レコードの番号の例

次ページの表は、レコードに付けられた番号について説明しています。表の各行は、レコードについての情報を示しています。行の順序は、レコードが出力フォームに表示される順序です。

- **データ列**: 各レコードのデータの内容で、例では人名を表しています。
- **レコード番号列**: レコードの絶対レコード番号です。この番号はRecord number関数によって求められます。
- **レコード位置番号列**: カレントセレクション中の位置を示す番号です。この番号はSelected record number関数によって求められます。
- **シーケンス番号列**: 各レコードに保存されたシーケンス番号です。この番号は、レコードが作成されたときにSequence number関数によって返される番号です。このシーケンス番号はフィールドに格納されています。

レコードの登録直後

最初の表は、入力された後のレコードを示しています。

- レコードのデフォルトの順序は、レコード番号順になります。
- レコード番号は0から始まります。
- レコード位置番号とシーケンス番号は、ともに1から始まります。

データ	レコード番号	レコード位置番号	シーケンス番号
Tess	0	1	1
Terri	1	2	2
Sabra	2	3	3
Sam	3	4	4
Lisa	4	5	5

Note: カレントセレクションを変更するコマンドを実行しても、レコードのデフォルト順序は変わりません。例えばデザインモードですべてを表示メニューを選択したり、ALL RECORDSコマンドを実行してもデフォルト順序は変わりません。

レコードのソート後

以下の表は、上の表と同じセレクションを名前でソートした後の状態を示しています。

- レコード番号は、各レコードに付けられたままです。
- レコード位置番号は、ソートされたセレクションの中の新しい位置を示しています。
- シーケンス番号は、各レコードが作成されたときにレコードに格納されているので、変わることはありません。

データ	レコード番号	レコード位置番号	シーケンス番号
Lisa	4	1	5
Sabra	2	2	3
Sam	3	3	4
Terri	1	4	2
Tess	0	5	1

レコードの削除後

以下の表はSamのレコードが削除されたセレクションの状態を示しています。

- レコード位置番号だけが変更されています。レコード位置番号はレコードが表示される順番を反映します。

データ	レコード番号	レコード位置番号	シーケンス番号
Lisa	4	1	5
Sabra	2	2	3
Terri	1	3	2
Tess	0	4	1

レコードの追加後

以下の表はLizという新しいレコードが追加されたセレクションの状態を示しています。

- 新しいレコードは、カレントセレクションの最後に加えられます。
- Samのレコード番号が新しいレコードLizに再使用されます。
- シーケンス番号は加算され続けます。

データ	レコード番号	レコード位置番号	シーケンス番号
Tess	0	1	1
Terri	1	2	2
Sabra	2	3	3
Lisa	4	4	5
Liz	3	5	6

セレクションの変更とソートの後

以下の表は、3つのレコードが選択され、さらにソートされたセレクションの状態を示しています。

- 各レコードのレコード位置番号だけが変更されています。

データ	レコード番号	レコード位置番号	シーケンス番号
Sabra	2	1	3
Liz	3	2	6
Terri	1	3	2

レコードスタックの使用

PUSH RECORDとPOP RECORDコマンドは、レコードをレコードスタックに積む (PUSH) 、またはレコードをレコードスタックから取り除き (POP) します。

各プロセスはテーブルごとに固有のレコードスタックを持っています。4Dは各レコードスタックを後入れ先出し法 (Last-In-First-Out:LIFO) で管理します。スタックの容量は、メモリの容量によって制限されます。

PUSH RECORDとPOP RECORDは慎重に使用してください。プッシュされた各レコードは、メモリの空き領域を使用します。数多くのレコードをプッシュしすぎると、メモリ不足やスタックがいっぱいの状態になります。

4Dは、メソッドを終了してメニューに戻った時点で、レコードスタックのポップされないレコード消去します。

PUSH RECORDとPOP RECORDは、データの入力中に同じテーブルの内容を調べるとき等に使用すると便利です。これを実行するためには、まずレコードをプッシュします。次に検索を行い、テーブルのレコードを検査 (例えばフィールドの内容を変数にコピーする等) します。最後にレコードを元の状態に戻すためにポップします。

レコード入力中に、複数のフィールドの重複しない値をチェックする必要がある場合、新しいコマンドであるSET QUERY DESTINATIONコマンドを使用してください。このコマンドにより、カレントレコードに入力したデータを一時的に保管しておく目的で、QUERYの前後にPUSH RECORDやPOP RECORDを呼び出す必要はなくなります。SET QUERY DESTINATIONを使用して、カレントセレクションやカレントレコードを変更しない検索を実行できます。

CREATE RECORD

CREATE RECORD {(aTable)}

引数	型		説明
aTable	テーブル	→	新規レコードを作成するテーブル, または 省略時、デフォルトテーブル

説明

CREATE RECORDは、aTableに対して新しい空のレコードを作成しますが、このレコードは表示されません。データ入力のために新しいレコードを作成して表示するにはADD RECORDを使用します。

CREATE RECORDは、レコードデータの割り当てをプログラミング言語で実行する場合に、ADD RECORDの代わりに使用します。新しく作成されたレコードはカレントレコードとなりますが、カレントセレクションは変わりません。

新しいレコードは、テーブルに対するSAVE RECORDコマンドを実行するまではメモリ上にのみ存在します。新しいカレントレコードが保存される前に(検索等によって) 変更されると、そのレコードは失われます。

例題

以下の例は、30日以上経過したデータをアーカイブします。これはアーカイブ用のテーブルに新しいレコードを作成することで行っています。アーカイブが終了すると、対象のレコードを[Accounts]テーブルから削除します:

```
  ` 30日経過したレコードを検索
QUERY([Accounts];[Accounts]Entered<(Current date-30))
For($vlRecord;1;Records in selection([Accounts])) ` レコードごとにループ
   CREATE RECORD([Archive]) ` 新しいアーカイブレコードを作成
   [Archive]Number:=[Account]Number ` アーカイブレコードにフィールドをコピー
   [Archive]Entered:=[Account]Entered
   [Archive]Amount:=[Account]Amount
   SAVE RECORD([Archive]) ` アーカイブレコードを保存
   NEXT RECORD([Accounts]) ` 次のaccountレコードをカレントレコードにする
End for
DELETE SELECTION([Accounts]) ` accountレコードを削除
```

DELETE RECORD

DELETE RECORD {(aTable)}

引数	型		説明
aTable	テーブル	→	カレントレコードを削除するテーブル, または 省略時、デフォルトテーブル

説明

DELETE RECORDは、aTableのカレントレコードを削除します。プロセスにaTableのカレントレコードが存在しない場合、DELETE RECORDコマンドは何も行いません。フォーム中では、このコマンドの代わりにレコード削除アクションを持つボタンを作成することができます。

Note: DELETE RECORDコマンドの実行前に、カレントレコードがメモリ上からアンロードにされている場合 (例えばUNLOAD RECORDコマンドの実行等)、削除実行後カレントセレクションは空になります。

レコードの削除は、一度実行すると元に戻すことはできません。

レコードが削除されると、そのレコード番号は新しいレコードが作成される際に再利用されます。したがって、データベースからレコードを削除することがある場合は、レコード番号をレコードの識別に使用しないでください。

例題

以下の例は1件の[employee]レコードを削除します。まずユーザにどのemployeeレコードを削除するのかを尋ね、[employee]レコードを検索し、見つかったレコードを削除します:

```
vFind:=Request("Employee ID to delete:") ` 従業員IDを要求
If(OK=1)
   QUERY([Employee];[Employee]ID =vFind) ` 従業員を検索
   DELETE RECORD([Employee]) ` 従業員を削除
End if
```

DISPLAY RECORD

DISPLAY RECORD {(aTable)}

引数	型		説明
aTable	テーブル	→	カレントレコードを表示するテーブル, または 省略時、デフォルトテーブル

説明

DISPLAY RECORDコマンドは、カレント入力フォームを使って、aTableのカレントレコードを表示します。レコードはイベントがウィンドウを更新するまでのみ表示されます。このイベントとは、ADD RECORDを実行する、または入力フォームへ戻る、メニューバーに戻ることです。DISPLAY RECORDは、カレントレコードが存在しない場合には何も行いません。

DISPLAY RECORDは、しばしばオリジナルの進捗メッセージを表示するために使用されます。また、自動スライドショーを作成するために使用されることもあります。

フォームメソッドが存在する場合は、On Loadイベントが生成されます。

警告: DISPLAY RECORDをWeb接続プロセスから呼び出さないでください。このコマンドは、Webブラウザクライアントマシン上ではなく、4D Webサーバマシン上で実行されるためです。

例題

以下の例は一連のレコードをスライドショーとして表示します:

```
ALL RECORDS([Demo]) ` レコードを選択
FORM SET INPUT([Demo];"Display") ` 表示に使用するフォームを選択
For($vlRecord;1;Records in selection([Demo])) ` レコード数だけループ
   DISPLAY RECORD([Demo]) ` レコードを表示
   DELAY PROCESS(Current process;180) ` 3秒間一時停止
   NEXT RECORD([Demo]) ` 次のレコードをカレントレコードにする
End for
```

DUPLICATE RECORD

DUPLICATE RECORD {(aTable)}

引数	型		説明
aTable	テーブル	→	カレントレコードを複製するテーブル, または 省略時、デフォルトテーブル

説明

DUPLICATE RECORDは、カレントレコードを複製して同じaTable内に新しいレコードを作成します。新しいレコードはカレントレコードとなります。カレントレコードが存在しない場合、DUPLICATE RECORDは何も行いません。新しいレコードを保存するにはSAVE RECORDを使用しなければなりません。

DUPLICATE RECORDはデータ入力中にも実行可能です。これにより現在表示しているレコードのコピーを作ることができます。複製元のレコードへの変更を保存するために、最初にSAVE RECORDを実行しておくことを忘れないでください。

互換性に関する注意: 4Dバージョン11より、このコマンドはサブテーブルをサポートしません。

GOTO RECORD

GOTO RECORD ({aTable ;} record)

引数	型	説明
aTable	テーブル	⇒レコードを移動するテーブル, または 省略時、デフォルトテーブル
record	倍長整数	⇒Record numberで返される番号

説明

GOTO RECORDは、aTable中の指定したレコードをカレントレコードとして選択します。record引数は、Record numberコマンドで求めることのできるレコード番号です。このコマンドを実行するとセレクションは選択されたレコード1件だけになります。

recordがデータベースの中で最も小さいレコード番号よりも小さい場合や、最も大きいレコード番号よりも大きい場合、4Dからレコード番号が範囲外である旨のエラーメッセージが表示されます。recordが削除されたレコードのレコード番号と等しい場合、4Dはエラー-10503を返し、セレクションは空になります。

例題

Record numberの例題参照

Is new record

Is new record {(aTable)} -> 戻り値

引数	型		説明
aTable	テーブル	→	レコードを検査するテーブル または 省略時、デフォルトテーブル
戻り値	ブール	←	True: レコードは未保存の新規レコード, そうでなければFalse

説明

Is new record コマンドは、カレントプロセス内で、指定されたaTableのカレントレコードが未保存の新規レコードの場合にTrueを返します。

互換性メモ: Record numberコマンドが-3を返すかどうかで同じ情報を得ることができます。
しかしこの目的ではRecord numberの代りにIs new recordを使用することを強くお勧めします。実際、Is new recordコマンドは4Dの将来のバージョンとのより優れた互換性を保証します。

4D Server: このコマンドは、On Validateフォームイベントにおいては、4Dローカルモードと4Dリモートモードで異なる値を返します。ローカルモードではFalse (レコードは既に作成されていると扱われるため) を返します。リモートモードではTrueを返します。なぜならば、レコードは4D Server上に作成されていますが、クライアントにこの情報はまだ通知されていないためです。

例題

下記の2つの方法は同一のものです。コードが4Dの将来のバージョンとの互換性を保つため、2番目の方法を強く推奨します:

```
If(Record number([Table])=-3) `非推奨
 ` ...
End if

If(Is new record([Table])) `強く推奨
 ` ...
End if
```

Is record loaded

Is record loaded {(aTable)} -> 戻り値

引数	型	説明
aTable	テーブル	→ レコードを検査するテーブル または 省略時、デフォルトテーブル
戻り値	ブール	← True: レコードはロードされている そうでなければFalse

説明

コマンドは、aTableのカレントレコードがカレントプロセス内にロードされていればTrueを返します。

例題

"次レコード"または"前レコード"の自動アクションを使用するかわりに、これら2つのボタン用にオブジェクトメソッドを書いて、これらの動作を向上させることができます。"次へ"ボタンは、カレントレコードがセレクションの最後のレコードであればセレクションの最初のレコードを表示し、"前へ"ボタンは、セレクションの最初のレコードであればセレクションの最後のレコードを表示します。

```
  ` "前へ" ボタンのオブジェクトメソッド （自動アクションなし）
 If(Form event=On Clicked)
    PREVIOUS RECORD([Group])
    If(Not(Is record loaded([Group])))
       GOTO SELECTED RECORD([Group];Records in selection([Group]))
  `セレクションの最後のレコードに移動
    End if
 End if
```

```
  ` "次へ" ボタンのオブジェクトメソッド （自動アクションなし）
 If(Form event=On Clicked)
    NEXT RECORD([Group])
    If(Not(Is record loaded([Group])))
       GOTO SELECTED RECORD([Groups];1)
  `セレクションの最初のレコードに移動
    End if
 End if
```

Modified record

Modified record {(aTable)} -> 戻り値

引数	型		説明
aTable	テーブル	→	カレントレコードが修正されているかテストするテーブル, または 省略時、デフォルトテーブル
戻り値	ブール	←	True: レコードは修正されている False: レコードは修正されていない

説明

Modified recordは、aTableのレコードが更新されたが保存されていない場合にTrueを返します。それ以外はFalseを返します。この関数は、保存する必要があるレコードかどうかを判定する場合に使用します。これは、入力フォーム上で次のレコードに移動する前にカレントレコードを保存するかどうかチェックする際に特に有効です。このコマンドは新規レコードについては、常にTrueを返します。

Note: Modifiedは、PUSH RECORDとPOP RECORDコマンド実行後は、常にTrueを返します。

例題

以下の例は、Modified recordの典型的な使用方法です:

```
If(Modified record([Customers]))
   SAVE RECORD([Customers])
End if
```

POP RECORD

POP RECORD {(aTable)}

引数	型		説明
aTable	テーブル	→	レコードをポップするテーブル, または 省略時、デフォルトテーブル

説明

POP RECORDは、aTableに属するレコードを、そのテーブルのレコードスタックからポップし、そのレコードをカレントレコードにします。

レコードをプッシュした後に、プッシュしたレコードを含まないようにカレントセレクションを変更した場合、レコードをポップしてもカレントレコードはカレントセレクションに含まれません。ポップしたレコードをカレントセレクションのレコードにするには、ONE RECORD SELECTを使用します。レコードを保存する前にレコードポインタを移動するようなコマンドを実行した場合、メモリ上のコピーを失います。

例題

以下の例はCustomers"テーブルのレコードをレコードスタックからポップします:

```
POP RECORD([Customers]) ` Pop customer's record onto stack
```

PUSH RECORD

PUSH RECORD {(aTable)}

引数	型		説明
aTable	テーブル	➡	レコードをプッシュするテーブル, または 省略時、デフォルトテーブル

説明

PUSH RECORDは、aTableのカレントレコード (それに付随するサブレコードも含めて) を、そのテーブルのレコードスタックにプッシュします。PUSH RECORDは、レコードがディスクに保存される前でも実行することができます。

変更可能 (アンロックされた) なレコードをプッシュした場合、ポップしてアンロードするまで、そのレコードはすべてのユーザやプロセスに対して変更不可 (ロック) のままです。

互換性に関する注意: 4Dバージョン11より、このコマンドはサブテーブルをサポートしません。

例題

以下の例は、[Customer]テーブルのレコードをレコードスタックにプッシュします:

```
PUSH RECORD([Customer])  ` 顧客レコードをスタックにプッシュする
```

Record number

Record number {(aTable)} -> 戻り値

引数	型		説明
aTable	テーブル	→	カレントレコードの番号を返すテーブル 省略時、デフォルトテーブル
戻り値	倍長整数	←	カレントレコード番号

説明

Record numberは、aTableのカレントレコードの物理レコード番号を返します。レコードポインタがカレントセレクションの前後にある場合等、カレントレコードがない場合、**Record number**は-1を返します。カレントレコードが保存されていない新しいレコードの場合、**Record number**は-3を返します。

レコード番号は変わることがあります。削除されたレコードのレコード番号は再利用されます。

4D Server: このコマンドは、On Validateフォームイベントにおいては、4Dローカルモードと4Dリモートモードで異なる値を返します。ローカルモードではレコード番号 (レコードは既に作成されていると扱われるため) を返します。リモートモードでは-3を返します。なぜならば、レコードは4D Server上に作成されていますが、クライアントにこの情報はまだ通知されていないためです。

注: レコードが作成中であるかをテストする目的では、Is new recordコマンドの利用をお勧めします。

例題

以下の例は、カレントレコードのレコード番号を変数に格納し、他に同じデータを持つレコードがないかを検索します:

```
$RecNum:=Record number([People]) ` レコード番号を取得
QUERY([People];[People]Last =[People]Last) ` 同じLast名を持つレコードを検索
  ` 検索件数を表示
ALERT("There are "+String(Records in selection([People]))+" with that name.")
GOTO RECORD([People];$RecNum) ` 元のレコードに戻る
```

Records in table

Records in table {(aTable)} -> 戻り値

引数	型		説明
aTable	テーブル	→	レコード数を返すテーブル, または 省略時、デフォルトテーブル
戻り値	倍長整数	↩	テーブル中の総レコード数

説明

Records in tableは、aTable中の総レコード数を返します。Records in selectionは、カレントセレクションのレコード数のみを返します。Records in tableがトランザクション内で使用される場合、トランザクション中に作成されたレコードが考慮に入れられます。

例題

以下の例は、テーブルのレコード数を表示します:

```
ALERT("There are "+String(Records in table([People]))+" records in the テーブル.")
```

SAVE RECORD

SAVE RECORD {(aTable)}

引数	型		説明
aTable	テーブル	→	カレントレコードを保存するテーブル, または 省略時、デフォルトテーブル

説明

SAVE RECORDは、カレントプロセスのaTableのカレントレコードを保存します。カレントレコードが存在しない場合、SAVE RECORDは何も行いません。

SAVE RECORDは、プログラムコードを使って新しく作成または修正したレコードを保存するために使用します。フォームでユーザが修正し確定したレコードは、SAVE RECORDで保存する必要はありません。ユーザによってフォーム中で修正されたレコードがキャンセルされた場合でも、SAVE RECORDで保存することができます。

レコード中のフィールドデータが変更されていない状態で SAVE RECORD コマンドを呼び出しても、コマンドはなにも行いません (トリガは呼び出されません)。

SAVE RECORDが必要とされる場合を次に示します:

- CREATE RECORDやDUPLICATE RECORDで作成した新しいレコードを保存する場合
- RECEIVE RECORDで取得したレコードを保存する場合
- メソッドによって修正したレコードを保存する場合
- ADD SUBRECORD、CREATE SUBRECORD、MODIFY SUBRECORDによって作成または修正したサブレコードを含むレコード保存する場合
- カレントレコードを変更するようなコマンドを実行する前に、データ入力途中で表示されているレコードを保存する場合
- データ入力途中でカレントレコードを保存する場合

受け入れられたフォームのOn ValidateイベントでSAVE RECORDを実行してはいけません。もし、これを実行すると、レコードが2回保存されてしまいます。

例題

以下の例はドキュメントからレコードを読み込むメソッドの一部です。このコードはレコードを受信し、この後受信が正常に行われると、レコードを保存します:

```
RECEIVE RECORD([Customers])  ` ディスクからレコードを受信
If(OK=1)  ` レコードを正しく受信したら…
   SAVE RECORD([Customers])  ` 保存する
End if
```

Sequence number

Sequence number {(aTable)} -> 戻り値

引数	型		説明
aTable	テーブル	→	シーケンス番号を求めるテーブル, または 省略時、デフォルトテーブル
戻り値	倍長整数	↩	シーケンス番号

説明

Sequence numberは、aTableの次のシーケンス番号を返します。シーケンス番号は、各テーブルにおいて固有のものです。この番号は、aTableに対して新しいレコードが追加されるたびに加算される、決して重複することのない番号です。番号は1から始まります。SET DATABASE PARAMETERコマンドを使用して、番号を変更することができます。

Sequence numberが有効なケース:

- 番号の増分値に1以上の数を使用する必要がある場合
- シーケンス番号を他のコードの一部に使用する場合（部品コードの一部に使用する場合等）

メソッドを使用してシーケンス番号を各レコードに格納するには、テーブル上に倍長整数型のフィールドを作成し、そのフィールドにシーケンス番号を代入します。

このコマンドから返されるtableのシーケンス番号は、ストラクチャインスペクタを使用してテーブルフィールドの**自動インクリメント**をチェックしたとき、あるいはフォームのフィールドにデフォルト値として#N記号を設定した場合に得られる番号と同じです。詳細は4D Design Referenceマニュアルを参照してください。

Note: 自動インクリメントはSQLのAUTO_INCREMENT属性で設定することもできます。

シーケンス番号が1以外の数値から始まる必要がある場合には、Sequence numberに対してその差を加算するだけで構いません。例えば、番号が1000から始まる必要がある場合には、以下のようなステートメントを使用します:

```
[Table1]SeqField :=Sequence number([Table1])+999
```

例題

以下の例は、フォームメソッドの一部です。このステートメントは、まずレコードが新しいものかどうか検査（請求書番号が空の文字列であるかどうかで判断）します。新しいレコードであれば、請求書番号を設定します。この請求書番号は、2つの情報から成り立っています。それは、シーケンス番号号とデータベースを開くときに入力された操作番号です。シーケンス番号を、5桁の文字列として扱います:

```
  ` 請求番号が空なら新たに作成する
If([Invoices]Invoice No="")
  ` 請求番号は5桁のシーケンス番号とオペレータIDからなる
   [Invoices]Invoice No:=String(Sequence number;"00000")+[Invoices]OpID
End if
```

レコードロック

- レコードのロック
- LOAD RECORD
- Locked
- LOCKED ATTRIBUTES
- READ ONLY
- Read only state
- READ WRITE
- UNLOAD RECORD

レコードのロック

4Dと4D Serverは、マルチユーザまたはマルチプロセスのコンフリクトを防ぐことによってマルチユーザデータベースを自動的に管理します。2人のユーザまたは2つのプロセスが、同時に同じレコードやオブジェクトを修正することはできませんが、2番目のユーザやプロセスはレコードやオブジェクトに読み込みのみのアクセスを得ることができます。

この章のマルチユーザコマンドを使用しなければならない状況がいくつかあります:

- プログラミング言語を使用してレコードを更新する。
- マルチユーザ環境でカスタムユーザインタフェースを使用する。
- トランザクション内で関連する変更を保存する。

マルチプロセスデータベースでコマンドを使用するときに注意すべき重要な概念が3つあります:

- 各テーブルは、読み込み専用または読み書き可能のどちらかに設定される。
- レコードは、ロードされた時点で他のプロセスに対しロック状態となり、アンロードされた時点でロック解除になる。
- ロックされたレコードは、更新することはできない。

この章では、マルチユーザデータベースで作業を実行する人を**ローカルユーザ**と呼びます。マルチユーザデータベースを使用する他の人を**他のユーザ**と呼びます。この節では、ローカルユーザの観点で説明します。またマルチプロセスの観点から、データベース上で処理を実行しているプロセスを**カレントプロセス**と呼びます。その他の実行中のプロセスは**他のプロセス**と呼びます。この章では、カレントプロセスの観点で説明します。

ロックされたレコード

ローカルユーザやカレントプロセスは、ロックされたレコードを更新できません。ロックされたレコードをロードすることは可能ですが、更新することはできません。他のユーザやプロセスが更新するためにレコードをロードした時点またはスタックした時点で、そのレコードはロックされます。レコードを更新しているユーザだけが、そのレコードをロックされていない状態で取り扱うことができます。その他のすべてのユーザやプロセスは、レコードがロック状態になるため更新することはできません。ロックされていない状態でレコードをロードするには、テーブルを必ず読み書き状態に設定しなければなりません。

読み込みのみと読み書き状態

データベース中の各テーブルは、データベースの各ユーザおよび各プロセスに対して読み込みのみ状態と読み書き状態のいずれかになっています。**読み込みのみ**とは、テーブルからレコードをロードすることはできるが更新することはできないという状態です。読み書きとは、テーブルのレコードをロード可能であり、他のユーザが先にそのレコードをロックしていない場合には更新することができる状態です。

テーブルの状態を変更すると、その変更は次からロードされるレコードに影響を及ぼすことに注意してください。テーブル状態を変更する際に既にロードされていたレコードは、状態変更によって影響を受けることはありません。

読み込みのみ状態

テーブルが読み込みのみ状態に設定されると、ロードしたレコードは常に更新不可にされます。つまり、レコードの表示と印刷を実行することはできますが、更新することはできません。

この読み込みのみ状態は、既存レコードの更新処理に対してのみ適用されることに注目してください。読み込みのみ状態は新規レコードの作成に影響を与えません。デザインモードのメニューやCREATE RECORD、ADD RECORDを使って読み込みのみテーブルにレコードを追加することができます。

4Dは、レコードに対する書き込み動作を必要としないコマンド利用時、テーブルを自動的に読み込みのみ状態にします。以下そのコマンドを示します:

- DISPLAY SELECTION
- DISTINCT VALUES
- EXPORT DIF
- EXPORT SYLK
- EXPORT TEXT
- GRAPH TABLE
- PRINT SELECTION
- PRINT LABEL
- QR REPORT
- SELECTION TO ARRAY
- SELECTION RANGE TO ARRAY

テーブルの現在の状態を知るには、Read only stateコマンドを使用します。

これらのコマンドを実行する前に、4Dはカレントプロセスにおけるテーブルのそのときの状態 (読み込みのみまたは読み書き) を保持します。コマンドを実行した後でテーブルの状態を元に戻します。

読み書き状態

読み書き状態のテーブルからロードされたレコードは、他のユーザがそのレコードを先にロックしていなければ更新可能になります。レコードが他のユーザによってロックされている場合にはレコードをロックされたレコードとしてロードできますが、更新はできません。

テーブルが読み書き状態に設定され、ロードしたレコードがロックされていない時に、レコードの更新が可能になります。

あるユーザが読み書き状態のテーブルからレコードをロードすると、他のユーザはレコードをロードできますが、更新することはできません。しかし他のユーザはデザインモード内での手動によるレコード追加、あるいはCREATE RECORDやADD RECORDを使ってのレコード追加をすることはできます。

データベースが開かれたりまたは新規プロセスが開始された時点で、すべてのテーブルはデフォルトで読み書き状態です。

テーブル状態の変更

READ ONLYおよびREAD WRITEコマンドを使って、テーブル状態を変更することができます。あるレコードを読み込みのみ状態または読み書き状態にするために任意テーブルの状態を変更したい場合は、そのレコードをロードする前にコマンドを実行する必要があります。既にロードされたレコードは、READ ONLYおよびREAD WRITEによって影響を受けることはありません。

各プロセス毎に、データベース内の各テーブルに対する独自の状態 (読み込みのみまたは読み書き) を持っています。

レコードのロード、更新、アンロード

ローカルユーザがレコードを更新するためには、テーブルが読み書き状態でかつ、ロードしたレコードがロックされていない状態でなければなりません。

NEXT RECORD, QUERY, ORDER BY, RELATE ONE等 のカレントレコードをロードするコマンドは、そのレコードをロックまたはロック解除状態にします。レコードは、テーブルのその時の状態 (読み込みのみまたは読み書き) とそのレコードの状態に応じてロードされます。自動リレートを使用するコマンドを使用した場合、リレート先テーブルからもレコードがロードされます。

テーブルが読み込みのみ状態になっている場合、そのテーブルからロードされたレコードはロックされています。他のプロセスから、ロックされたレコー ドの保存および削除はできません。読み込みのみの状態を使用することをお勧めします。この状態は他のユーザに対しレコードのロード、更新、保存を可能にし ます。

テーブルが読み書き状態になっている場合、そのテーブルからロードされたレコードは他のユーザが先にロックしていない限り更新可能になります。更新可能なレコードは、更新して保存することができます。テーブルは、レコードをロードする前に読み書き状態にしなければなりません。

レコードを修正する場合、Locked関数を使って、他のユーザがレコードをロックしていないかどうかを調べます。レコードがロックされている (LockedがTrueを返す) 場合、レコードをLOAD RECORDコマンドでロードし、そのレコードがロックされているかどうかを再び調べます。レコードがロック解除される (LockedがFalseを返す) までこの処理を繰り返します。

レコードの更新が完了したら、UNLOAD RECORDを使ってそのレコードを 解放 (他のユーザに対しロック解除) しなければなりません。レコードがアンロードされないと、他のカレントレコードが選択されるまで、すべての他のユーザに対してロックされた状態のままにな ります。テーブルのカレントレコードを変えると、前のカレントレコードは自動的にロック解除されます。カレントレコードを変更しない場合は、UNLOAD RECORDコマンドを明示的に呼び出す必要があります。これは既存レコードの場合だけであり、新しいレコードを作成する場合、そのレコードが属するテーブルの状態に関係なく保存することができます。

Note: トランザクションの中でUNLOAD RECORDコマンドを使用した場合、トランザクションを管理するプロセス中でのみカレントレコードがアンロードされます。他のプロセスから見た場合、レコードはロックされたままとなり、トランザクションを完了 (または取消し) されるまで維持されます。

LOCKED ATTRIBUTESコマンドを使用すると、レコードをロックしているユーザやプロセスを知ることができます。

ロック解除されたレコードをロードするためのループ

以下の例は、アンロックされたレコードをロードするための最も単純なループ処理を示しています:

```
READ WRITE([Customers]) ` テーブルを読み書きに設定
Repeat ` レコードがアンロックされるまでループ
   LOAD RECORD([Customers]) ` レコードをロードし、他のプロセスに対しロックする
   If(Locked([Customers])) ` まだロックされていれば
      DELAY PROCESS(Current process;5) ` 絶え間ないロード指示を避ける
   End if
Until(Not(Locked([Customers])))
 ` レコードに処理を行う
READ ONLY([Customers]) ` テーブルを読み込みのみにする
```

このループ処理は、レコードがロック解除されるまで繰り返されます。

このようなループ処理は、ユーザがループが終了するまで待たされるため、レコードが他のユーザにロックされる可能性がほとんどない場合にのみ使用することができます。したがって、メソッドからのみレコードを更新するような場合以外では使用することはできません。

以下の例は先のループを使用してアンロックされたレコードをロードし、更新を行っています:

```
READ WRITE([Inventory]) ` テーブルを読み書きに設定
Repeat ` レコードがアンロックされるまでループ
   LOAD RECORD([Inventory]) ` レコードをロードし、他のプロセスに対しロックする
   If(Locked([Inventory])) ` まだロックされていれば
      DELAY PROCESS(Current process;5) ` 絶え間ないロード指示を避ける
   End if
Until(Not(Locked([Inventory])))
[Inventory]Part Qty:=[Inventory]Part Qty-1 ` レコードを更新
SAVE RECORD([Inventory]) ` レコードを保存
UNLOAD RECORD([Inventory]) ` 他のユーザに対しアンロックする
READ ONLY([Inventory])
```

MODIFY RECORDコマンドはレコードがロックされている場合には、自動的にそれをユーザに通知し、レコードが更新されることを防ぎます。以下の例では最初にLocked関数を使用してレコードの状態を調べることで、この自動的な通知を避けています。レコードがロックされている場合には、ユーザが処理を中断できるようにします。

以下の例は、[commands]テーブルのカレントレコードがロックされているかどうかを調べます。ロックされている場合は、プロセスがメソッドによって1秒間延期されます。この技法は、マルチユーザやマルチプロセスの場合両方に用いることができます:

```
Repeat
   READ ONLY([Commands]) ` 今は読み書きである必要はない
   QUERY([Commands])
 ` 検索が終了すると、いくつかのレコードが返される
   If((OK=1) & (Records in selection([Commands])>0))
      READ WRITE([Commands]) ` テーブルを読み書きにする
      LOAD RECORD([Commands])
      While(Locked([Commands]) & (OK=1)) `レコードがロックされていたら
 ` レコードがアンロックされるまでループ
 ` ロックしているのは誰か?
         LOCKED ATTRIBUTES([Commands];$Process;$User;$Machine;$Name)
         If($Process=-1) ` レコードは削除済み?
            ALERT("The record has been deleted in the meantime.")
            OK:=0
         Else
            If($User="") ` シングルユーザモード
               $User:="you"
            End if
            CONFIRM("The record is already used by "+$User+" in the "+$Name+" Process.")
            If(OK=1) ` If you want to wait for a few seconds
               DELAY PROCESS(Current process;60) ` 1秒待つ
               LOAD RECORD([Commands]) ` レコードのロードを試行
            End if
         End if
      End while
      If(OK=1) ` レコードがアンロックされている
         MODIFY RECORD([Commands]) ` レコードを更新
         UNLOAD RECORD([Commands])
      End if
      READ ONLY([Commands]) ` 読み込みのみに戻す
      OK:=1
   End if
Until(OK=0)
```

マルチユーザまたはマルチプロセス環境でのコマンドの使用

いくつかのコマンドは、レコードがロックされていることを検知した時点で、特定の処理を実行します。これらのコマンドは、レコードがロックされていない場合には、通常どおりの処理を遂行します。

以下に、レコードがロックされていることを検知した場合の各コマンドの処理を示します。

- MODIFY RECORD: レコードが使用されていることを示すダイアログボックスを表示します。レコードは表示されず、ユーザはレコードを修正できません。デザインモードでは、レコードは読み込みのみ状態で表示されます。
- MODIFY SELECTION: ユーザがレコードをダブルクリックして修正しようとした場合を除いて通常どおりの処理を行います。MODIFY SELECTIONはレコードが使用されていることを示すダイアログボックスを表示し、読み込みのみ状態でレコードのアクセスを許可します。
- APPLY TO SELECTION: ロックされたレコードをロードしますが、そのレコードを更新しません。APPLY TO SELECTIONは、特別な処置を行わずにテーブルからレコードを読み取ります。ロックされたレコードを検知すると、コマンドはそのレコードをLockedSetシステムセットに格納します。
- DELETE SELECTION: ロックされたレコードを削除せずにスキップします。コマンドはロックされたレコードを見使えると、それをLockedSetシステムセットに格納します。
- DELETE RECORD: レコードがロックされている場合、このコマンドは何も行いません。エラーも返しません。このコマンドを実行する前にレコードがアンロックされていることを確認する必要があります。
- SAVE RECORD: レコードがロックされている場合、このコマンドは何も行いません。エラーも返しません。このコマンドを実行する前にレコードがアンロックされていることを確認する必要があります。
- ARRAY TO SELECTION: ロックされたレコードは保存しません。ロックされたレコードを検知すると、コマンドはそのレコードをLockedSetシステムセットに格納します。
- GOTO RECORD: マルチユーザ/マルチプロセスデータベース上では、他のユーザがレコードを削除、または追加することができます。したがって、レコード番号が変更される場 合があるため、マルチユーザデータベース上でレコード番号を使用して直接レコードを参照する場合には、十分注意してください。
- : セットに使用される情報を他のユーザやプロセスが変更する可能性があるため、セットの使用には細心の注意が必要です。

LOAD RECORD

LOAD RECORD {(aTable)}

引数	型		説明
aTable	テーブル	→	レコードをロードするテーブル, または 省略時、デフォルトテーブル

説明

LOAD RECORDは、aTableのカレントレコードをロードします。カレントレコードが存在しない場合、LOAD RECORDは何も行いません。
レコードがロードされたら、Lockedコマンドを使用してレコードが更新可能か調べることができます:

- aTableが読み込みのみ状態ならば、LockedコマンドはTrueを返し、レコードを修正することはできません。
- aTableが読み書き状態でも、レコードが既にロックされている場合は、レコードは読み込みのみ状態になり、そのレコードを修正することはできません。
- aTableが読み書き状態でかつレコードがロックされていない場合、カレントプロセス内でレコードを修正することができます。その際LockedコマンドはTrueを返します。その他のすべてのユーザとプロセスに対してTrueを返します。

Note: READ ONLYコマンドの後にLOAD RECORDを実行すると、UNLOAD RECORDコマンドを使用しなくても、レコードは自動的にアンロードされた後ロードされます。

通常QUERY、NEXT RECORD、PREVIOUS RECORD等のコマンドは自動的にカレントレコードをロードするため、LOAD RECORDコマンドを使用する必要はありません。

マルチユーザ環境やマルチプロセス環境において既存レコードを修正するには、レコードが属するテーブルに読み書き状態でアクセスしなければなりません。レコードがロックされており、ロードされていない場合、時間をおいて、再度LOAD RECORDを実行する必要があります。このためには、ループ内でLOAD RECORDを使用し、レコードが読み書き状態になるまで待機できます。

Tip: LOAD RECORDコマンドを使用し、入力フォームにおいてカレントレコードを再ロードすることができます。変更中のデータはすべて以前の値で置き換えられます。この場合、LOAD RECORDコマンドは、いわば一般的な入力データの取消しとして振る舞います。

Locked

Locked {(aTable)} -> 戻り値

引数	型	説明
aTable	テーブル	→ ロックを検証するレコードが属するテーブル, または 省略時、デフォルトテーブル
戻り値	ブール	↩ TRUE: レコードはロックされている FALSE: レコードはロックされていない

説明

Lockedは、aTableのカレントレコードがロックされているかを調べます。このコマンドを使用してレコードがロックされているかどうかを調べた後、レコードが開放されるまで待機するか、処理をスキップするかの選択をユーザに与える等の適切な処理を行ってください。

LockedがTrueを返す場合、レコードは他のユーザまたはプロセスによりロックされているか、カレントプロセスでスタックされており、レコードを保存することはできません。この場合には、LOAD RECORDコマンドを使用して、**Locked**がFalseを返すまでレコードのロードを繰り返します。

LockedがFalseを返す場合、レコードはアンロックされています。これはレコードが他のすべてのユーザに対してロックされることを意味します。ローカルユーザまたはカレントプロセスだけがレコードを修正、保存できます。レコードを修正するには、テーブルが読み書き状態でなければなりません。

削除されたレコードをロードしようとすると、**Locked**はTrueを返し続けます。存在しないレコードをこれ以上待機しないために、LOCKED ATTRIBUTESコマンドを使用します。レコードが削除されている場合、LOCKED ATTRIBUTESコマンドはプロセス引数に-1を返します。

Note: aTableにカレントレコードが存在しない場合、言い換えればRecord numberが-1を返す時、**Locked**はFalseを返します。

トランザクション処理の実行中、レコードがロックされているかどうかを調べるためにLOAD RECORDと**Locked**がしばしば使用しされます。レコードがロックされている場合、トランザクション処理をキャンセルするのが一般的です。

LOCKED ATTRIBUTES

LOCKED ATTRIBUTES ({aTable ;} process ; 4Duser ; sessionUser ; processName)

引数	型	説明
aTable	テーブル	→レコードロックをテストするテーブル, または 省略時、デフォルトテーブル
process	倍長整数	←プロセス参照番号
4Duser	文字	←4Dユーザ名
sessionUser	文字	←ワークセッションを開いているユーザ
processName	文字	←プロセス名

説明

LOCKED ATTRIBUTESは、レコードをロックしたユーザやプロセスに関する情報を返します。process, 4Duser, sessionUser, そしてprocessName変数にはそれぞれサーバマシン上のプロセス番号、4Dアプリケーションのユーザ名、システムユーザ名、そしてプロセス名が返されます。レコードがロックされている場合、これらの情報を使用してカスタムダイアログ内でユーザに警告できます。

レコードがロックされていない場合、processは0を返し、4Duser、sessionUser、およびprocessNameは空の文字列を返します。読み込み状態でロードしようとしたレコードが削除されている場合には、processは-1を返し、4Duser、sessionUser、およびprocessNameは空の文字列を返します。

4Duser引数には、たとえユーザ名が空白であっても、4Dパスワードシステムのユーザ名が返されます。パスワードシステムがない場合は、"Designer"が返されます。

sessionUser引数はクライアントマシンでセッションを開いたユーザ名に対応します。この名前は特に、4D Serverの管理ウィンドウに、開かれたプロセスごとに表示されます。

READ ONLY

READ ONLY {(aTable | *)}

引数	型		説明
aTable | *	テーブル, 演算子	→	読み込みのみにするテーブル, または *: すべてのテーブル, または 省略時: デフォルトテーブル

説明

READ ONLYは、コマンドが呼び出されたプロセス内のaTableの状態を読み込みのみに変更します。このコマンドを実行した後でロードしたレコードはすべてロックされ、変更することはできません。* を指定すると、すべてのテーブルが読み込みのみに変更されます。

レコードを修正する必要のない場合に、READ ONLYを使用します。

Note: このコマンドは、コマンドが実行された時点からさかのぼって適用されることはありません。レコードはロードされた時点におけるテーブルの読み書き状態に従ってロードされます。読み書き可のテーブルから、読み込みのみ状態でレコードをロードするには、先にテーブルを読み込みのみ状態に変更する必要があります。

Read only state

Read only state {(aTable)} -> 戻り値

引数	型		説明
aTable	テーブル	→	読み込みのみ状態を調べるテーブル, または 省略時はデフォルトテーブル
戻り値	ブール	↩	TRUE: テーブルへのアクセスは読み込みのみ FALSE: テーブルへのアクセスは読み書き可

説明

このコマンドは、コマンドが呼び出されるプロセス内のaTableの状態が読み込み専用かどうかを調べます。aTableが読み込み専用であれば、Trueを返します。aTableが読み書き可であれば、Falseを返します。

例題

以下の例は[Invoice]テーブルの状態を判断するものです。[Invoice]テーブルの状態が読み込み専用であれば、読み書き可に設定し、カレントレコードを再度ロードします。

```
If(Read only state([Invoice]))
   READ WRITE([Invoice])
   LOAD RECORD([Invoice])
End if
```

Note: ユーザがレコードを修正できるようにするために、請求書レコードを再ロードします。すでに読み込み専用でロードされたレコードは、書き込み可能状態で再ロードされるまで、変更不可のままです。

READ WRITE

READ WRITE {(aTable | *)}

引数	型		説明
aTable | *	テーブル, 演算子	→	読み書き可にするテーブル, または *: すべてのテーブル, または 省略時: デフォルトテーブル

説明

READ WRITEは、このコマンドを呼び出したプロセス内のaTableを読み書き可に変更します。*を指定すると、すべてのテーブルが読み書き可に変更されます。

READ WRITEコマンドを呼び出した後、レコードがロードされると、他のユーザがそのレコードをロックしていない場合には更新可能になります。このコマンドはすでにロードされたレコードの状態は変更しません。実行後にロードしたレコードに対してのみ適用されます。

すべてのテーブルのデフォルト状態は、読み書き可です。

READ WRITEコマンドは、レコードを修正しその結果を保存しなければならないときに使用します。またレコードを修正しない場合でも、他のユーザに対してレコードをロックする必要のあるときにもこのコマンドを使用します。テーブルを読み書き可モードに設定することにより、他のユーザによるそのテーブルに対する編集を防ぐことができます。ただし、他のユーザは新規レコードの作成は行えます。

Note: このコマンドは、コマンドが実行された時点からさかのぼって適用されることはありません。レコードはロードされた時点におけるテーブルの読み書き設定状態に従ってロードされます。読み込みのみのテーブルから、読み書き可状態でレコードをロードするには、先にテーブルを読み書き可状態に変更する必要があります。

UNLOAD RECORD

UNLOAD RECORD {(aTable)}

引数	型		説明
aTable	テーブル	→	レコードをアンロードするテーブル, または 省略時、デフォルトテーブル

説明

UNLOAD RECORDは、aTableのカレントレコードをアンロードします。

ローカルユーザがレコード修正可能なとき (他のユーザにとっては修正不可の場合)、UNLOAD RECORDコマンドは他のユーザに対してレコードをロック解除します。

UNLOAD RECORDはメモリからレコードをアンロードしますが、そのレコードはカレントレコードのままです。他のレコードがカレントレコードになると、前のカレントレコードは自動的にアンロードされ、他のユーザに対してロック解除されます。レコードの修正が終わり、そのレコードを自分自身のカレントレコードとしたままで、他のユーザから使えるようにしたい場合は常にこのコマンドを実行します。

レコードにピクチャフィールドや4D Write等の外部ドキュメントなど大きなデータが含まれている場合、修正が必要でないかぎりそのカレントレコードをメモリ内に保持したくないかもしれません。こういう場合、UNLOAD RECORDコマンドを使用して、メモリ内にレコードを持たずにカレントレコードを保持できるようにします。そのレコードによって占有されていたメモリを解放できますが、そのフィールドの値にアクセスすることはできません。後にレコードの値へのアクセスが必要となった場合、LOAD RECORDコマンドを使用します。

Note: トランザクションの中でUNLOAD RECORDコマンドを使用した場合、トランザクションを管理するプロセス内でのみカレントレコードがアンロードされます。他のプロセスから見た場合、レコードはロックされたままとなり、トランザクションを完了 (または取消し) されるまで維持されます。

入力制御

- EDIT ITEM
- FILTER KEYSTROKE
- Get edited text
- GET HIGHLIGHT
- GOTO OBJECT
- HIGHLIGHT TEXT
- Keystroke
- REJECT

EDIT ITEM

EDIT ITEM ({* ;} object {; item})

引数	型		説明
*	演算子	→	指定した場合オブジェクト名（文字列）省略するとテーブルまたは変数
object	フォームオブジェクト	→	オブジェクト名 (*が指定されている場合) または テーブルまたは変数 (*が省略された場合)
item	倍長整数	→	項目番号

説明

EDIT ITEM コマンドはobject 引数で指定された項目、または指定された配列やリスト中item で指定された項目を編集状態にします。
これは選択された項目を編集したり、内容を入れ替える新しい文字を入力できるようになるということを意味します。

オプションの* 引数を渡すと、object 引数がオブジェクト名であることを指定したことになります。この場合objectには文字列を渡します。この引数を渡さない場合、object 引数にはテーブルまたは変数を渡します。この場合文字列ではなくテーブルまたは変数参照を渡します。

このコマンドは以下の入力可能オブジェクトに適用されます:

- 階層リスト
- リストボックス
- サブフォーム (この場合objectにはサブフォームオブジェクト名を渡します)
- MODIFY SELECTION または DISPLAY SELECTIONで表示されたリストフォーム

コマンドがリストでない入力可オブジェクトに使用された場合、GOTO OBJECT コマンドと同様に動作します。
このコマンドはリストや配列が空、あるいは非表示の際には何も行いません。またリストや配列が入力不可の場合、このコマンドは単に指定された項目を選択し、入力状態にはしません。リストボックスについては、列が入力を許可しない場合 (チェックボックスやドロップダウンリストの入力のみ許可される場合)、指定された要素がフォーカスを得ます。

オプションのitem 引数には編集モードにする、階層リストの場合項目位置を、複数選択モードでのリストボックスやリストフォーム、サブフォームでは行番号を指定します。この引数を渡さない場合、このコマンドはobjectのカレント項目に適用されます。カレントの項目がない場合、object の最初の項目が編集モードになります。

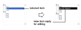
Note:

- サブフォームとリストフォームでは、コマンドは指定された行の、最初の入力順のフィールドを編集モードにします。
- 階層表示モードのリストボックスでは、ターゲットの項目が折りたたまれた階層レベルに属する場合、このレベル (およびすべての親レベル) は自動で展開され、項目が表示されます。

例題 1

このコマンドは特に階層リストに新規項目を作成する際に有用です。このコマンドが呼び出されると、最後に追加あるいは挿入された項目が自動で編集モードになります。
以下のコードは既存のリストに新規項目を挿入するボタンメソッドです。デフォルトの"New_item"テキストが自動で編集可能状態になります:

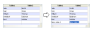
```
vlUniqueRef:=vlUniqueRef+1
INSERT IN LIST(hList;*;"New_item";vlUniqueRef)
EDIT ITEM(*;"MyList")
```



例題 2

変数名"Array1" と "Array2"が指定された2つの列を持つリストボックスがあります。以下の例題では2つの配列に新規項目を挿入し、Array2を編集モードにします:

```
$vlRowNum:=Size of array(Array1)+1
LISTBOX INSERT ROWS(*;"MyListBox";$vlRowNum)
Array1{$vlRowNum}:="New value 1"
Array2{$vlRowNum}:="New value 2"
EDIT ITEM(Array2;$vlRowNum)
```



FILTER KEYSTROKE

FILTER KEYSTROKE (filteredChar)
引数　　　型　　説明
filteredChar 文字⇒フィルタされたキーストローク文字、または 空文字の場合キーストロークをキャンセル

説明

FILTER KEYSTROKE は、ユーザがフィールドや入力可エリアに入力した文字を、filteredChar に渡した文字列の最初の文字で置き換えます。

空の文字を渡すと、キーストロークはキャンセルされ無視されます。

通常FILTER KEYSTROKEはOn Before Keystrokeフォームイベントを処理するフォームメソッドあるいはオブジェクトメソッドで呼び出します。キーストロークイベントを検知するには、Form eventコマンドを使用します。実際のキーストロークを得るにはKeystrokeコマンドを使用します。

重要: FILTER KEYSTROKEコマンドはユーザが入 力した他の文字をキャンセルしたり置き換えたりすることを可能にします。他方特定のキーストロークに対し複数の文字を入力したい場合、スクリーン上に表示 されているテキストは、編集中のデータソースフィールドや変数の値にまだなっていないことに注意してください。データソースフィールドや変数にはデータ入 力が確定された後に値が代入されます。変数へのデータ入力を途中で取得し、この値を変更して、入力エリアに代入するのは開発者に任されています (この節の例題を参照)。

FILTER KEYSTROKE は以下の目的で使用できます:

- カスタマイズされた方法で文字をフィルタする
- データ入力フィルタで実現できない方法でデータ入力のフィルタを行う
- ダイナミックなルックアップやタイプアヘッドの実装

警告: FILTER KEYSTROKEの後にKeystrokeコマンドを呼び出すと、実際に入力された文字の代わりにこのコマンドに渡された文字が返されます。

例題 1

以下のコードを使用すると:

```
  ` myObject enterable area object method
 Case of
    :(Form event=On Load)
       myObject:=""
    :(Form event=On Before Keystroke)
       If(Position(Keystroke;"0123456789")>0)
          FILTER KEYSTROKE("*")
       End if
 End case
```

myObject エリアに入力されたすべての数字がアスタリスクに変換されます。

例題 2

以下のコードはパスワード入力エリアを実装します。入力された文字はスクリーン上にランダムな文字で表示されます:

```
  ` vsPassword enterable area object method
 Case of
    :(Form event=On Load)
       vsPassword:=""
       vsActualPassword:=""
    :(Form event=On Before Keystroke)
       Handle keystroke(->vsPassword;->vsActualPassword)
       If(Position(Keystroke;Char(Backspace)+Char(Left Arrow Key)+
          Char(Right Arrow Key)+Char(Up Arrow Key)+Char(Down Arrow Key))=0)
          FILTER KEYSTROKE(Char(65+(Random%26)))
       End if
 End case
```

データ入力が確定されると、ユーザが実際に入力したデータはvsActualPasswordに格納されています。

Note: Handle keystroke メソッドはKeystrokeの節で紹介されています。

例題 3

アプリケーションに文章を入力できる欄があります。またアプリケーションには辞書テーブルがあります。テキストエリア編集中に、テキストエリア中で選択された文字に基づき、辞書の内容を取り出して挿入したいとします。これを行うには2つの方法があります:
- キーを割り当てたボタンを用意する
- テキストエリアの編集中、特別なキーストロークを検知する

この例題ではHelpキーを使用して、2つ目のソリューションを実装します。

上で説明した通り、テキストエリアの編集中、このエリアのデータソースにはデータ入力を確定した後に入力された値が代入されます。テキストエリア編集中、辞書の内容を取り出してこのエリアに挿入するために、データ入力をシャドウする必要があります。最初の2つの引数として、この入力エリアとシャドウ 用の変数へのポインタを、3番目の引数として禁止文字を渡します。キーストロークがどのように取り扱われるにしても、メソッドは元のキーストロークを返し ます。禁止文字は入力エリアに挿入されず、特別に扱いたい文字のことです。

```
  ` Shadow keystroke project method
  ` Shadow keystroke ( Pointer ; Pointer ; String) -> String
  ` Shadow keystroke ( -> srcArea ; -> curValue ; Filter ) -> Old keystroke
 C_STRING(1;$0)
 C_POINTER($1;$2)
 C_TEXT($vtNewValue)
 C_STRING(255;$3)
  ` 元のキーストロークを返す
 $0:=Keystroke
  ` 入力エリア中で選択されている文字範囲を取得する
 GET HIGHLIGHT($1->;$vlStart;$vlEnd)
  ` 現在値を処理する
 $vtNewValue:=$2->
  ` 押されたキーまたは入力された文字に基づき、
  ` 適切な処理を行う
 Case of
  ` (Delete) キーが押された
    :(Character code($0)=Backspace)
  ` 選択されている文字あるいはカーソルの左の文字を削除
       $vtNewValue:=Delete text($vtNewValue;$vlStart;$vlEnd)
  ` 矢印キーが押されたら
  ` 何も行わずキーストロークを受け入れる
    :(Character code($0)=Left Arrow Key)
    :(Character code($0)=Right Arrow Key)
    :(Character code($0)=Up Arrow Key)
    :(Character code($0)=Down Arrow Key)

  ` 入力を許可する文字が入力された
    :(Position($0;$3)=0)
       $vtNewValue:=Insert text($vtNewValue;$vlStart;$vlEnd;$0)
    Else
  ` 入力を許可しない
       FILTER KEYSTROKE("")
 End case
  ` 次のキーストローク処理のために値を返す
 $2->:=$vtNewValue
```

このメソッドは以下のサブメソッドを使用します:

```
  ` Delete text プロジェクトメソッド
  ` Delete text ( String; Long ; Long ) -> String
  ` Delete text ( -> Text ; SelStart ; SelEnd ) -> New text
 C_TEXT($0;$1)
 C_LONGINT($2;$3)
 $0:=Substring($1;1;$2-1-Num($2=$3))+Substring($1;$3)
  ` Insert text プロジェクトメソッド
  ` Insert text ( String; Long ; Long ; String) -> String
  ` Insert text ( -> srcText ; SelStart ; SelEnd ; Text to insert ) -> New text
 C_TEXT($0;$1;$4)
 C_LONGINT($2;$3)
 $0:=$1
 If($2#$3)
    $0:=Substring($0;1;$2-1)+$4+Substring($0;$3)
 Else
    Caes of
 :($2<=1)
    $0:=$4+$0
 :($2>Length($0))
    $0:=$0+$4
 Else
    $0:=Substring($0;1;$2-1)+$4+Substring($0;$2)
End case
End if
```

プロジェクトにこれらのメソッドを実装したのち、以下のように使用することができます:

```
  ` vsDescription 入力可エリアのプロジェクトメソッド
 Case of
    :(Form event=On Load)
       vsDescription:=""
       vsShadowDescription:=""
  ` 特別なキーとして使用する禁止文字リストを作成
  ` ( この例題ではHelpキーをフィルタする)
       vsSpecialKeys:=Char(HelpKey)
    :(Form event=On Before Keystroke)
       $vsKey:=Shadow keystroke(->vsDescription;->vsShadowDescription;vsSpecialKeys)
       Caes of
    :(Character code($vsKey)=Help Key)
  ` Help キーが押された時の処理
  ` この例題では辞書を検索し、挿入する
       LOOKUP DICTIONARY(->vsDescription;->vsShadowDescription)
 End case
End case
```

LOOKUP DICTIONARYプロジェクトメソッドは以下のようなものです。このメソッドの目的はシャドウの変数を使用して検索を行い、編集中のエリアに再代入を行うことです:

```
  ` LOOKUP DICTIONARY プロジェクトメソッド
  ` LOOKUP DICTIONARY ( Pointer ; Pointer )
  ` LOOKUP DICTIONARY ( -> Enterable Area ; ->ShadowVariable )

 C_POINTER($1;$2)
 C_LONGINT($vlStart;$vlEnd)

  ` 入力エリアの選択範囲を取得
 GET HIGHLIGHT($1->;$vlStart;$vlEnd)
  ` 選択されたテキストあるいはカーソルの左側の単語を取得
 $vtHighlightedText:=Get highlighted text($2->;$vlStart;$vlEnd)
  ` 検索すべきものがあるか
 If($vtHighlightedText#"")
  ` テキストセレクションがカーソルなら
  ` 選択はテキストカーソルの前の単語から始まる
    If($vlStart=$vlEnd)
       $vlStart:=$vlStart-Length($vtHighlightedText)
    End if
  ` 最初に有効な辞書エントリを検索
    QUERY([Dictionary];[Dictionary]Entry=$vtHighlightedText+"@")
  ` 存在するか
    If(Records in selection([Dictionary])>0)
  ` 存在すればシャドウテキストに挿入する
       $2->:=Insert text($2->;$vlStart;$vlEnd;[Dictionary]Entry)
  ` シャドウテキストを編集中のエリアにコピーする
       $1->:=$2->
  ` 挿入した辞書入力の後に選択をセットする
       $vlEnd:=$vlStart+Length([Dictionary]Entry)
       HIGHLIGHT TEXT(vsComments;$vlEnd;$vlEnd)
    Else
  ` 辞書に対応するエントリがなかった
       BEEP
    End if
 Else
  ` ハイライトされたテキストがない
    BEEP
 End if
```

Get highlighted text メソッド:

```
  ` Get highlighted text プロジェクトメソッド
  ` Get highlighted text ( String; Long ; Long ) -> String
  ` Get highlighted text ( Text ; SelStart ; SelEnd ) -> highlighted text
 C_TEXT($0;$1)
 C_LONGINT($2;$3)
 If($2<$3)
    $0:=Substring($1;$2;$3-$2)
 Else
    $0:=""
    $2:=$2-1
    Repeat
       If($2>0)
          If(Position($1[[$2]];" ,.!?:;()-_")=0)
             $0:=$1?$2?+$0
             $2:=$2-1
          Else
             $2:=0
          End if
       End if
    Until($2=0)
 End if
```

Get edited text

Get edited text -> 戻り値

引数	型	説明
戻り値	テキスト	↩ 編集中のテキスト

説明

Get edited textコマンドは、主にOn After Keystrokeフォームイベントで入力中のテキストを取得するために使用します。またOn Before Keystrokeフォームイベントと共に使用することもできます。詳細はForm eventの説明を参照してください。

Note: 4D バージョン6.5で実装された新しいフォームイベントであるOn After Keystrokeに対応して、従来のイベントである*On Keystroke*はOn Before Keystrokeと呼ばれるようになりました。

フォームオブジェクト内でテキストを入力しないコンテキストでは、このコマンドは空の文字列を返します。

例題 1

以下のメソッドは、入力される文字を自動で大文字に変換します:

```
If(Form event=On After Keystroke)
   [Trips]Agencies:=Uppercase(Get edited text)
End if
```

例題 2

以下はテキストフィールドへの文字入力をオンザフライで処理する例です。これは入力中の文のすべての単語を、"Words"という他のテキストフィールドに置くというアイデアに基づきます。これを実行するには、フィールドのオブジェクトメソッド内に下記のコードを記述します:

```
If(Form event=On After Keystroke)
   $RealTimeEntry:=Get edited text
   PLATFORM PROPERTIES($platform)
   If($platform#3) ` Mac OS
      Repeat
         $DecomposedSentence:=Replace string($RealTimeEntry;Char(32);Char(13))
      Until(Position(" ";$DecomposedSentence)=0)
   Else ` Windows
      Repeat
         $DecomposedSentence:=Replace string($RealTimeEntry;Char(32);Char(13)+Char(10))
      Until(Position(" ";$DecomposedSentence)=0)
   End if
   []Words:=$DecomposedSentence
End if
```

Note: この例題は、単語がスペース(Char(32))によって区切られていると仮定しているため、完全なものではありません。完全な解決法としては、すべての単語を抽出するように他のフィルタを付加する必要があります(カンマ、セミコロン、アポストロフィー等の区切り)。

GET HIGHLIGHT

GET HIGHLIGHT ({* ;} object ; startSel ; endSel)

引数	型	説明
*	演算子	⇒ 指定時: objectはオブジェクト名 (文字列)、省略時: objectはフィールドまたは変数
object	フィールド, 変数, フォームオブジェクト	⇒ オブジェクト名 (*指定時)、またはフィールドや変数 (*省略時)
startSel	倍長整数	⇐ 反転表示された先頭位置
endSel	倍長整数	⇐ 反転表示された最終位置

説明

GET HIGHLIGHTコマンドは、object中で現在反転表示されているテキストを検出するために使用します。

オプションの * 引数を渡すとobject引数はオブジェクト名 (文字列) です。* 引数を渡さないとobject引数はフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フォーム上のフィールドや変数のみ) を渡します。

注: このコマンドをサブフォームのリストフォーム内にあるフィールドに対して使用することはできません。

テキストは、ユーザによる指定やHIGHLIGHT TEXTコマンドの実行で反転させることができます。

引数 startSel は反転表示された文字の最初の位置を返します。
引数 endSel は反転表示された文字の最後の位置に1を加えて返します。

startSel と endSel が同じである場合、挿入ポインタはstartSel によって指定された文字の前にあります。テキストは選択されておらず、文字は反転されていません。

object引数で指定されたオブジェクトがフォーム中で見つからなかった場合、コマンドは startSel に-1を、endSel に-2を返します。

例題 1

以下の例題を使用して、フィールド [Products]Comments から反転表示された部分を検出します。

```
GET HIGHLIGHT([Products]Comments;vFirst;vLast)
If(vFirst<vLast)
   ALERT("The selected text is: "+Substring([Products]Comments;vFirst;vLast-vFirst))
End if
```

例題 2

FILTER KEYSTROKE コマンドの例題を参照してください。

例題 3

ハイライトされたテキストのスタイルを変更する:

```
GET HIGHLIGHT(*;"myText";$startsel,$endsel)
OBJECT SET STYLED TEXT ATTRIBUTE(*;"myText";$startsel,$endsel;Attribute underline style;1;Attribute bold style;1)
```

GOTO OBJECT

GOTO OBJECT ({* ;} object)

引数	型		説明
*	演算子	→	指定した場合 = objectはオブジェクト名 (文字列) 省略した場合 = object はフィールドまたは変数参照
object	フィールド, 変数	→	オブジェクト名 (* を指定した場合) または フィールドまたは変数 (* を省略した場合)

説明

GOTO OBJECTコマンドは、フォームのアクティブエリアとしてデータ入力オブジェクトobject を選択するために使用します。これはユーザがフィールドや変数をクリックしたりタブでフォーカスを移動したりするのと同じです。

オプションの * 引数を渡した場合、文字列でオブジェクト名を object に渡します。オプションの * 引数を省略した場合、object にはフィールドまたは変数渡します。この場合文字列ではなく、フィールまたは変数参照を渡します。オブジェクト名に関する詳細はの節を参照してください。

カレントのフォームのオブジェクトからフォーカスを外すには、このコマンドに空のオブジェクト名を渡します (例題 2参照)。

GOTO OBJECT コマンドをサブフォームのコンテキストで利用できます。コマンドがサブフォームから呼び出されると、まずサブフォーム内でオブジェクトを探し、そしてなにも見つからなければ親フォーム内を検索します。

例題 1

GOTO OBJECTコマンドは以下のように使用します:

```
GOTO OBJECT([People]Name)  ` フィールド参照
GOTO OBJECT(*;"AgeArea")  ` オブジェクト名
```

例題 2

フォーカスを外す:

```
GOTO OBJECT(*;"")
```

例題 3

REJECTコマンドの例題参照

HIGHLIGHT TEXT

HIGHLIGHT TEXT ({* ;} object ; startSel ; endSel)

引数	型		説明
*	演算子	⇒	指定時: objectはオブジェクト名 (文字列)、省略時: objectはフィールドまたは変数
object	フィールド, 変数, フォームオブジェクト	⇒	オブジェクト名 (*指定時)、またはフィールドや変数 (*省略時)
startSel	倍長整数	⇒	反転表示の先頭位置
endSel	倍長整数	⇒	反転表示の最終位置

説明

HIGHLIGHT TEXTコマンドは、object 内にあるテキストの一部を反転表示します。

オプションの * 引数を渡すとobject引数はオブジェクト名 (文字列) です。* 引数を渡さないとobject引数はフィールドまたは変数です。この場合、文字列ではなくフィールドまたは変数参照 (フォーム上のフィールドや変数のみ) を渡します。

object が現在編集中のオブジェクトでない場合、フォーカスはこのエリアにセットされます。

注: このコマンドをサブフォームのフィールドに対して使用することはできません。

startSel は、反転表示する先頭文字の位置です。lastSel は、反転表示する最終文字の位置に1を加えた値です。startSel と lastSel が同じ場合は、挿入ポインタが startSel で指定された文字の前に置かれ、文字は全く反転表示されません。

lastSel が object の文字数より大きい場合、startSel からテキストの最終までのすべての文字を反転表示します。

例題 1

以下の例題を使用して、入力可能なフィールド [Products]Comments のすべての文字を選択します。

```
HIGHLIGHT TEXT([Products]Comments;1;Length([Products]Comments)+1)
```

例題 2

以下の例題を使用して、挿入ポインタを入力可能なフィールド [Products]Comments の始めに移動します。

```
HIGHLIGHT TEXT([Products]Comments;1;1)
```

例題 3

以下の例題を使用して、挿入ポインタを入力可能なフィールド [Products]Comments の終わりに移動します。

```
$vLen:=Length([Products]Comments)+1
HIGHLIGHT TEXT([Products]Comments;$vLen;$vLen)
```

例題 4

FILTER KEYSTROKE コマンドの例題を参照してください。

Keystroke

Keystroke -> 戻り値

引数	型	説明
戻り値	文字	ユーザが入力した文字

説明

Keystroke はユーザがフィールドや入力可能エリアに入力した文字を返します。

通常、KeystrokeはOn Before Keystrokeフォームイベントを処理するフォーム/オブジェクトメソッドで使用します。キーストロークイベントを検知するには、Form eventコマンドを使用します。

ユーザが実際に入力した文字を置き換えるには、FILTER KEYSTROKEコマンドを使用します。

Note: Keystroke 関数はサブフォームでは動作しません。

重要: 編集中の入力可エリアの現在値や新しく入力された値を使用して逐次処理を行う場合、スクリーン上に表示されているテキストは、編集中のデータソース フィールドや変数の値にまだなっていないことに注意してください。データソースフィールドや変数には、タブで他のエリアに移動したりボタンをクリックする などして、データ入力が確定された後に値が代入されます。変数へのデータ入力を途中で変数に取得し、この値を処理することは開発者に任されています。特定 の処理を行うために現在のテキスト値を所得しなければならない場合、この作業を行う必要があります。なおGet edited textも使用できます。

Keystroke コマンドは以下の目的で使用します:

- カスタマイズされた方法で文字をフィルタする
- データ入力フィルタで実現できないフィルタを実装する
- ダイナミックなルックアップやタイプアヘッドを実装する

例題 1

FILTER KEYSTROKEコマンドの例を参照

例題 2

On Before Keystroke イベントを処理する際、(カーソルがある) 現在のテキストエリアの編集中の値を扱います。このエリアのデータソース (フィールドまたは変数) の将来の値ではありません。Handle keystroke プロジェクトメソッドはテキストエリアのデータ入力を2つ目の変数にコピーします。この変数を使用して、入力中の文字に基づく処理を行うことができます。 第1引数にはエリアのデータソースへのポインタを渡します。2番目の引数にはコピー先の変数へのポインタを渡します。メソッドはコピー先変数にテキストエ リアの新しい値を返し、そして最後に入力された文字が挿入された値と異なっていればTrueを返します。

```
  ` Handle keystroke プロジェクトメソッド
  ` Handle keystroke ( Pointer ; Pointer ) -> Boolean
  ` Handle keystroke ( -> srcArea ; -> curValue ) -> Is new value

 C_POINTER($1;$2)
 C_TEXT($vtNewValue)

  ` 入力エリアで現在選択されている範囲を取得
 GET HIGHLIGHT($1->;$vlStart;$vlEnd)
  ` 現在の値を処理する
 $vtNewValue:=$2->
  ` 押されたキーや入力された文字に基づき、
  ` 適切な動作を行う
 Case of

  ` Backspace (Delete) キーが押されたら
    :(Character code(Keystroke)=Backspace)
  ` 選択されたテキストまたはテキストカーソルの左の文字を削除
       $vtNewValue:=Substring($vtNewValue;1;$vlStart-1-Num($vlStart=$vlEnd))
       +Substring($vtNewValue;$vlEnd)

  ` 入力可能な文字が入力されたら
    :(Position(Keystroke;"abcdefghjiklmnopqrstuvwxyz -0123456789")>0)
       If($vlStart#$vlEnd)
  ` 1つ以上の文字が選択されていれば、入力された文字で置き換える
          $vtNewValue:=Substring($vtNewValue;1;$vlStart-1)
          +Keystroke+Substring($vtNewValue;$vlEnd)
       Else
  ` カーソルが置かれているだけなら
          Caes of
  ` カーソルがテキストの先頭にある
       :($vlStart<=1)
  ` 文字をテキストの先頭に挿入
          $vtNewValue:=Keystroke+$vtNewValue
  ` カーソルがテキストの最後にある
       :($vlStart>=Length($vtNewValue))
  ` 文字をテキストに追加
          $vtNewValue:=$vtNewValue+Keystroke
       Else
  ` カーソルがテキスト中にある。新しい文字を挿入する
          $vtNewValue:=Substring($vtNewValue;1;$vlStart-1)+Keystroke
          +Substring($vtNewValue;$vlStart)
    End case
 End if

  ` 矢印キーが押された
  ` 何もせず、キーストロークを受け入れる
:(Character code(Keystroke)=Left Arrow Key)
:(Character code(Keystroke)=Right Arrow Key)
:(Character code(Keystroke)=Up Arrow Key)
:(Character code(Keystroke)=Down Arrow Key)
  `
Else
  ` 文字、数字、ダッシュ以外のキーストロークを受け付けない
 FILTER KEYSTROKE("")
End case
  ` 値が異なっているか?
$0:=($vtNewValue#$2->)
  ` 次回のキーストローク処理のために値を返す
$2->:=$vtNewValue
```

このプロジェクトメソッドがプロジェクトに追加されたら、以下のように使用できます:

```
  ` myObject enterable area オブジェクトメソッド
 Case of
    :(Form event=On Load)
       MyObject:=""
       MyShadowObject:=""
    :(Form event=On Before Keystroke)
       If(Handle keystroke(->MyObject;->MyShadowObject))
  ` MyShadowObjectに格納された値に基づき、適切なアクションを行う
       End if
 End case
```

以下のフォームを見てみましょう:

このフォームには以下のオブジェクトが置かれています: vsLookup入力エリア、vsMessage入力不可エリア、asLookupスクロールエリア。vsLookupに文字を入力する間、オブジェクトメソッドが[US Zip Codes]テーブルの検索を実行し、年の最初の文字を入力すると、USの都市が検索されます。

vsLookup オブジェクトメソッドは以下のようになります:

```
  ` vsLookup 入力エリアオブジェクトメソッド
 Case of
    :(Form event=On Load)
       vsLookup:=""
       vsResult:=""
       vsMessage:="検索する都市の最初の文字を入力してください。"
       CLEAR VARIABLE(asLookup)
    :(Form event=On Before Keystroke)
       If(Handle keystroke(->vsLookup;->vsResult))
          If(vsResult#"")
             QUERY([US Zip Codes];[US Zip Codes]City=vsResult+"@")
             MESSAGES OFF
             DISTINCT VALUES([US Zip Codes]City;asLookup)
             MESSAGES ON
             $vlResult:=Size of array(asLookup)
             Case of
                :($vlResult=0)
                   vsMessage:="都市が見つかりませんでした。"
                :($vlResult=1)
                   vsMessage:="1つの都市が見つかりました。"
                Else
                   vsMessage:=String($vlResult)+" cities found."
             End case
          Else
             DELETE FROM ARRAY(asLookup;1;Size of array(asLookup))
             vsMessage:="検索する都市の最初の文字を入力してください。"
          End if
       End if
 End case
```

フォームを実行:

4Dのプロセス間通信を使用することで、レコードを編集するプロセスと通信するフローティングウィンドウに同様の機能を実装することができます。

REJECT

REJECT {(aField)}

引数	型		説明
aField	フィールド	➡	入力を拒否するフィールド

説明

REJECT には2つの形式があります。第1の形式は、引数がありません。これは、データ入力全体を取り消し、ユーザは強制的にフォーム上にとどまります。第2の形式は、aField だけを取り消し、ユーザは強制的にそのフィールド上にとどまります。

Note: このコマンドを使用する前に、組み込みのデータ検証ツールの利用を考慮すべきです。

REJECTコマンドの第1の形式では、完全でないレコードをユーザが受け入れないようにします。REJECTを使用しなくても同じ効果を得ることができます。フィールドが正しく入力された後、テンキー側の"enter"キーをショートカットとした「動作なし」ボタンのオブジェクトメソッドでACCEPTコマンドとCANCELコマンドを使用してレコードの受け入れや取り消しを行います。REJECTコマンドの第1の形式よりも、上記の2番目の手法を利用することをお勧めします。

第1の形式を使用する場合には、REJECTコマンドを実行してユーザがレコードを受け入れないようにしますが、通常これはレコードが完全ではなかったり、不正確な入力が行われたために行います。ユーザがレコードを受け入れようとすると、REJECTコマンドの実行によりレコードが受け入れられません。そしてこのレコードはフォーム上に表示されたままになります。したがって、ユーザはレコードが受け入れられるか、あるいは取り消されるまでデータの入力を続行しなければなりません。

この形式のREJECTコマンドを実行するのに最適な場所は、テンキー側の"enter"キーをショートカットとして割り当てた登録ボタンのオブジェクトメソッドです。この場合のデータのチェックは、レコードが受け入れられた場合にのみ実行されます。ユーザは"enter"キーを押して、データをチェックしないで済ませることはできません。

REJECTコマンドの第2の形式は、引数fieldを渡します。カーソルは、フィールドエリアにとどまります。この形式は、ユーザが正しい値を入力するように強制します。この場合、フィールドの修正の直後にREJECTコマンドを使用しなければなりません。Modified 関数を使用して、修正されたかどうかを判定します。データ入力エリアのオブジェクトメソッドでREJECTコマンドを使用することもできます。このコマンドは、サブフォームエリア上のフィールドでは動作しません。

REJECTコマンドのいずれかの形式をフォームメソッドまたはオブジェクトメソッドに記述しなければなりません。あるテーブルのサブフォームの詳細フォームにREJECTコマンドを使用する場合は、詳細フォームのフォームメソッドまたはオブジェクトメソッドに記述します。

取り消されたフィールドのデータを選択するために、HIGHLIGHT TEXTコマンドを使用することができます。

例題 1

以下の例題は銀行のトランザクション記録です。これは一番目の形式のREJECTを受け入れボタンで使用する方法を示しています。Enterキーがこのボタンのショートカットとして設定されています。つまりユーザがレコードを受け入れるためにEnterキーを押したとしても、ボタンのオブジェクトメソッドが実行されます。トランザクションが小切手であれば、小切手番号が必須です。小切手番号がなければれレコード保存は拒否されます:

```
Case of
   :(([Operation]Transaction="Check") & ([Operation]Check Number="")) ` If it is a check with no number...
      ALERT("Please fill in the check number.") ` Alert the user
      REJECT ` Reject the entry
      GOTO OBJECT([Operation]Check Number) ` Go to the check number field
End case
```

例題 2

以下の例は[Employees]Salaryフィールドのオブジェクトメソッドの一部です。このオブジェクトメソッドは、[Employees]Salaryフィールドを調べて1万ドル以下の場合には取り消しします。「フォーム」エディタでフィールドの最小値を指定しても、同じ処理を実現することができます:

```
If([Employees]Salary<10000)
   ALERT("給与は$10,000を超えなければなりません。")
   REJECT([Employees]Salary)
End if
```

割込

- ABORT
- ASSERT
- Asserted
- FILTER EVENT
- Get assert enabled
- GET LAST ERROR STACK
- Method called on error
- Method called on event
- ON ERR CALL
- ON EVENT CALL
- SET ASSERT ENABLED

ABORT

ABORT
このコマンドは引数を必要としません

説明

ABORT コマンドは、ON ERR CALLコマンドでインストールされたエラー処理プロジェクトメソッド内で使用します。

エラー処理プロジェクトメソッドが存在しない場合、エラーが発生すると(例えばデータベースエンジンエラー)、4Dは標準のエラーダイアログボックスを表示し、コードの実行が中断されます。実行しているコードにより、次のようになります:

- オブジェクトメソッド、フォームメソッド(あるいはフォームメソッドやオブジェクトメソッドからコールされたプロジェクトメソッド)の場合、現在表示されているフォームに制御が戻ります。
- メニューから呼び出されたメソッドの場合、メニューバーまたは現在表示されているフォームに制御が戻ります。
- プロセスのマスターメソッドの場合、プロセスは終了します。
- データの読み込み、または書き出し処理により直接的あるいは間接的に呼び出されたメソッドの場合、処理は中止されます。順次クエリや並び替え処理に関しても同様です。
- その他

エラー処理メソッドを使用してエラーを検知する場合、4Dは標準のエラーダイアログボックスの表示もコードの実行の中断も行いません。この代りに、4Dはエラー処理プロジェクトメソッドを呼び出し、エラーを発生したメソッドの次のコード行から実行を再開します。

プログラムで処理できるエラーもあります。例えば、インポート処理中にデータベースエンジンからの重複値エラーを検出した場合、このエラーを"カバー"して、インポート処理を続けることができます。しかし、対処できないエラーや"カバー"してはいけないエラーもあります。この場合、エラー処理プロジェクトメソッドからABORTコマンドを呼び出して、実行を中止する必要があります。

以前のバージョンに関する注意

ABORT コマンドはエラー処理プロジェクトメソッド内でのみ使用されるようになっていますが、4Dコミュニティには実行を中断するために他のプロジェクトメソッド で使用しているメンバーも存在します。これが動作するというのは単に副次的な効果に過ぎません。このコマンドをエラー処理メソッド以外のメソッド内で使用することはお勧めできません。

ASSERT

ASSERT (boolExpression {; messageText})

引数	型		説明
boolExpression	ブール	⇒	ブール式
messageText	テキスト	⇒	エラーメッセージテキスト

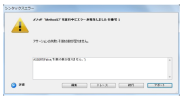
説明

ASSERT コマンドを使用してメソッドコードにアサーションを置くことができます。

アサーションはコードに挿入された指示命令で、コード実行中の例外を検知するために使用します。ある時点において式の検証をしたときにTrueであれば正常であり、そうでなければ例外が発生したことになります。アサーションはなにより、発生するはずのないケースを検知するために使用します。主にプログラミングバグを検知するために使用します。SET ASSERT ENABLEDコマンドを使用して、(例えばバージョンにより) アプリケーションのすべてのアサーションを全体として有効にしたり無効にしたりできます。

プログラミングにおけるアサーションについての詳細は、Wikipediaの関連情報をご覧ください: http://ja.wikipedia.org/wiki/表明

ASSERTコマンドは引数として渡されたブール式を評価します。式がTrueであればなにも起こりません。Falseの場合、コマンドはエラー -10518を生成し、"アサーション違反:"のあとにアサーションのテキストを表示します。



メソッド "Method17" を実行中にエラー が発生しました 行番号 1

アサーションの失敗: 引数の数が足りません。

```
ASSERT(False;"引数の数が足りま せん。")
```

ON ERR CALLコマンドを使用してインストールしたメソッドを使用して、例えばログファイルなどに記録を行うなどのために、エラーをとらえることができます。

コマンドはオプションで2番目の引数を受け入れます。これはブール式がFalseのときに表示されるテキストメッセージを変更します。

例題 1

レコードに対する処理を実行する前に、開発者はレコードが正しく読み/書きモードでロードされたかを確認する必要があります:

```
READ WRITE([Table 1])
LOAD RECORD([Table 1])
ASSERT(Not(Locked([Table 1])))
  // レコードがロックされていると -10518 エラーが生成される
```

例題 2

アサーションはプロジェクトメソッドに渡された引数をテストして、異常な値を検知するために使用できます。以下の例題では、カスタム警告メッセージが表示されます。

```
  // $1に渡された名前に基づき、クライアントの番号を返す
C_TEXT($1) // クライアントの名前
ASSERT($1#"";"クライアント名が空です")
  // このケースでは空の名前は異常な値です
  // アサーションがfalseの場合、以下がエラーダイアログに表示されます:
  //"アサーション違反: クライアント名が空です"
```

Asserted

Asserted (boolExpression {; messageText}) -> 戻り値

引数	型		説明
boolExpression	ブール	→	ブール式
messageText	テキスト	→	エラーメッセージテキスト
戻り値	ブール	←	boolExpressionの評価結果

説明

Asserted コマンドはASSERTコマンドと同様の処理を行います。1つの違いは、このコマンドはboolExpression引数の評価結果を戻り値として返すことです。このため、条件の評価としてアサーションを使用できます (例題参照)。アサーションの処理とこのコマンドの引数に関する詳細情報はASSERTコマンドの説明を参照してください。

Assertedはブール式を引数として受け入れ、この式の評価結果を返します。アサーションが有効で式がFalseの場合 (SET ASSERT ENABLEDコマンド参照)、ASSERTと同様エラー-10518が生成されます。アサーションが無効にされていると、Assertedはエラー生成することなしに、渡された式の結果を返します。

例題

式の評価部にアサーションを挿入する:

```
READ WRITE([Table 1])
LOAD RECORD([Table 1])
If(Asserted(Not(Locked([Table 1]))))
  // このコードはレコードがロックされているとエラー-10518を生成する
    ...
End if
```

FILTER EVENT

FILTER EVENT
このコマンドは引数を必要としません

説明

FILTER EVENT コマンドは、ON EVENT CALLコマンドでインストールされたイベント処理プロジェクトメソッドから呼び出されます。

イベント処理メソッドでFILTER EVENTを呼び出すと、カレントイベントが4Dに渡されなくなります。

このコマンドを使用すると、イベントキューからカレントイベント（クリック、キー入力）を取り除くことができます。したがって、4Dはイベント処理プロジェクトメソッド内で発生したイベントに対してそれ以上の処理は行いません。

警告：FILTER EVENTコマンドを呼び出すだけのイベント処理メソッドを作成しないようにしてください。そのようにするとすべてのイベントが4Dから無視されるためです。FILTER EVENTコマンドだけのイベント処理メソッドがある場合には、Ctrl+Shift+Alt+Backspace (Windows) または Command-Option-Shift-Control-Backspace (Macintosh) キーを押します。これにより、On Event Callプロセスがイベントをまったく受け取らない通常のプロセスに切り替わります。

特別なケース：フォームがDISPLAY SELECTION や MODIFY SELECTIONで表示されているとき、FILTER EVENT コマンドを標準の出力フォームメソッドで使用できます。この特別なケースでは、FILTER EVENTコマンドを使用してレコード上でのダブルクリックをフィルタすることができます（また、この方法でページモードでのレコードオープン以外の動作を実行します）。これを行うには、出力フォームメソッドに次の行を追加します：

```
If(Form event=On Double Clicked)
   FILTER EVENT
   ... `ダブルクリックを処理する
End if
```

例題

ON EVENT CALLコマンドの例題を参照.

Get assert enabled

Get assert enabled -> 戻り値

引数	型		説明
戻り値	ブール	↩	True = アサーションは有効 False = アサーションは無効

説明

Get assert enabled コマンドは カレントプロセスでアサーションが有効か無効かによってTrueまたはFalseを返します。アサーションについての詳細はASSERTコマンドの説明を参照してください。

デフォルトでアサーションは有効ですが、SET ASSERT ENABLEDコマンドを使用して無効にできます。

GET LAST ERROR STACK

GET LAST ERROR STACK (codesArray ; intCompArray ; textArray)

引数	型		説明
codesArray	倍長整数配列	⇦	エラー番号
intCompArray	文字配列	⇦	内部コンポーネントコード
textArray	文字配列	⇦	エラーテキスト

説明

GET LAST ERROR STACK コマンドは、4Dアプリケーションの現在のエラースタックに関する情報を返します。4Dのステートメントがエラーを起こしている場合、現在のエラースタックには生成された一連のエラーとともにエラーの説明が含まれます。例えば"ディスクがいっぱいです"タイプのエラーはファイルの書き込みエラーの原因となり、さらにレコード保存コマンドのエラーを起こします。この場合スタックには3つのエラーが含まれます。最後に実行された4Dステートメントがエラーを生成していない場合、エラースタックは空です。

この汎用コマンドは、4Dアプリケーションで発生するかもしれないすべてのタイプのエラーを処理するために使用できます。

Note: しかし、ODBCソースで生成されたエラーに関する詳細な情報を取得するためには、SQL GET LAST ERROR コマンドを使用する必要があります。

このコマンドはON ERR CALL コマンドでインストールされたエラー処理メソッドの中で呼び出さなければなりません。

情報は3つの同期された配列に返されます:

- codesArray: この配列は生成されたエラーコードのリストを受け取ります。
- intCompArray: この配列はそれぞれのエラーに関連する内部コンポーネントのコードが返されます。
- textArray: この配列にはそれぞれのエラーのテキストが返されます。

エラーコードとそのテキストのリストはエラーコードテーマで提供されます。

Method called on error

Method called on error -> 戻り値

引数	型		説明
戻り値	文字	←	エラー時に呼び出されるメソッド名

説明

Method called on error コマンドは、ON ERR CALLコマンドによりカレントプロセスにインストールされたメソッド名を返します。

メソッドがインストールされていない場合、空の文字列が返されます。

例題

このコマンドはコンポーネントでとくに有用です。エラー処理メソッドを一時的に変更し、後で復元することができます:

```
$methCurrent:=Method called on error
ON ERR CALL("NewMethod")
 ` ドキュメントを開くことができなければエラーが生成される
$ref:=Open document("MyDocument")
 ` 前のエラー処理メソッドに戻す
ON ERR CALL($methCurrent)
```

Method called on event

Method called on event -> 戻り値

引数	型		説明
戻り値	文字	↩	イベント発生時に呼び出されるメソッド名

説明

Method called on event コマンドは、ON EVENT CALLコマンドでインストールされたメソッド名を返します。

インストールされたメソッドが存在しない場合は、空の文字列を返します。

ON ERR CALL

ON ERR CALL (errorMethod)

引数	型		説明
errorMethod	文字	→	実行されるエラーメソッド, または 空の文字列でエラーのトラップ停止

説明

ON ERR CALL コマンドは、エラー検知用のメソッドとしてerrorMethodで渡した名前のプロジェクトメソッドをインストールします。このプロジェクトメソッドは**エラー処理メソッド**または**エラーキャッチメソッド**と呼ばれます。

このコマンドのスコープはカレントプロセスです。1つのプロセスには1度に1つのエラー処理メソッドだけを使用できますが、異なるプロセス間では、異なるエラー処理メソッドを持つことができます。

エラーの検知を中止するには、errorMethodに空の文字列を指定して再度ON ERR CALLコマンドをコールします。

エラー処理プロジェクトメソッドのインストール後は、エラーが発生するたびに4Dがこのメソッドを呼び出します。

エラーはシステム変数Errorの値で判別します。このシステム変数にはエラーコードが納められます。このマニュアルの付録に エラーコード が記載されています。詳細はシンタックスエラー (1 -> 81)を参照してください。システム変数Errorの値はエラー処理メソッド内のみで有効です。エラーの原因となったメソッド内でこのエラーコードが必要であれば、システム変数Errorを独自のプロセス変数にコピーしてください。またError methodとError lineシステム変数にはそれぞれエラーが発生したメソッドの名前とその行番号が設定されます (Error, Error method, Error line参照)。

エラー処理メソッドは適切な方法でエラーを管理、またはユーザに対してエラーメッセージを表示します。エラーは次のようなものから生成されます:

- 4Dデータベースエンジン: 例えば重複不可フィールドに既存の値を保存しようとした場合。
- 4D環境: 例えば配列に割り当てるために十分なメモリがない場合。
- データベースが稼働しているOS: 例えばディスクに空きがなかったり、I/Oエラーの場合。

実行を中断するには、ABORTコマンドを使用できます。エラー処理メソッドでABORTコマンドを使用しない場合、4Dは割り込みをかけたメソッドに制御を戻し、メソッドの実行を続けます。エラーをリカバーできないときにABORT コマンドを使用します。

エラー処理メソッド自体でエラーが発生した場合は、4Dがエラー管理を引き継ぎます。したがって、エラー処理メソッドでエラーが発生しないように十分注意してください。また、エラー処理メソッドではON ERR CALLコマンドを使用することはできません。

ON ERR CALLは通常On startupデータベースメソッドから呼び出され、このアプリケーションのエラーを処理します。またON ERR CALLはメソッドの開始時に置かれて、そのメソッド特有のエラーを処理します。

例題 1

次のプロジェクトメソッドは、引数で渡された名前のドキュメンを作成します。ドキュメントが作成できない場合、このプロジェクトメソッドは0またはエラーコードを返します:

```
  ` Create doc プロジェクトメソッド
  ` Create doc ( 文字列 ; ポインタ) -> 倍長整数
  ` Create doc ( DocName ; ->DocRef ) -> Error code result

gError:=0
ON ERR CALL("IO ERROR HANDLER")
$2->:=Create document($1)
ON ERR CALL("")
$0:=gError
```

IO ERROR HANDLERプロジェクトメソッドは以下のようになります:

```
  ` IO ERROR HANDLER project method
gError:=Error ` エラーコードをプロセス変数にコピー
```

現在実行中のメソッド内でエラーコードの結果を取得するために、プロセス変数gErrorを使用している点に注意してください。データベースにこれらのメソッドを作成したら、次のようなコードを使用します:

```
  ` ...
C_TIME(vhDocRef)
$vlErrCode:=Create doc($vsDocumentName;->vhDocRef)
If($vlErrCode=0)
  `...
   CLOSE DOCUMENT($vlErrCode)
Else
   ALERT("ドキュメントを作成できませんでした, I/O error "+String($vlErrCode))
End if
```

例題 2

配列とメモリの例題参照

例題 3

複雑な一連の処理を実装中に、各種サブルーチンで異なるエラー処理メソッドが必要となる場合があります。プロセスごとにいちどに1つのエラー処理メソッドしか持つことができないため、次の2通りの方法から対応策を選択することになります:
- ON ERR CALLコマンドを呼び出すたびに現在のエラー処理メソッドを保持する。または
- プロセス配列変数を使用し(この例ではasErrorMethod)、エラー処理メソッドとプロジェクトメソッド(この例ではON ERROR CALL)を"積み上げ"て、エラー処理メソッドのインストールとクリアを行う。

プロセスの実行を開始する時点で配列を初期化する必要があります:

```
  ` プロセスメソッドの最初に配列の初期化をするのを忘れないように。
ARRAY STRING(63;asErrorMethod;0)
```

これはカスタマイズしたON ERROR CALLメソッドです:

```
  ` ON ERROR CALL project method
  ` ON ERROR CALL { ( 文字列 ) }
  ` ON ERROR CALL { ( Method Name ) }

C_STRING(63;$1;$ErrorMethod)
C_LONGINT($vlElem)

If(Count parameters>0)
   $ErrorMethod:=$1
Else
   $ErrorMethod:=""
End if

If($ErrorMethod#"")
   C_LONGINT(gError)
   gError:=0
   $vlElem:=1+Size of array(asErrorMethod)
   INSERT IN ARRAY(asErrorMethod;$vlElem)
   asErrorMethod{$vlElem}:=$1
   ON ERR CALL($1)
Else
   ON ERR CALL("")
   $vlElem:=Size of array(asErrorMethod)
   If($vlElem>0)
      DELETE FROM ARRAY(asErrorMethod;$vlElem)
      If($vlElem>1)
         ON ERR CALL(asErrorMethod{$vlElem-1})
      End if
   End if
End if
```

次のように呼び出します:

```
gError:=0
ON ERROR CALL("IO ERRORS") ` IO ERRORS エラー処理メソッドをインストール
  ` ...
ON ERROR CALL("ALL ERRORS") ` ALL ERRORS エラー処理メソッドをインストール
  ` ...
ON ERROR CALL ` ALL ERRORS エラー処理メソッドをアンインストールして、IO ERRORSを再インストール
  ` ...
ON ERROR CALL ` IO ERRORS エラー処理メソッドをアンインストール
  ` ...
```

例題 4

次のエラー処理メソッドはユーザによる割り込みを無視します:

```
  ` SHOW ONLY ERRORS project method
If(Error#1006)
   ALERT("The error "+String(Error)+" occurred.")
End if
```

ON EVENT CALL

ON EVENT CALL (eventMethod {; processName})

引数	型	説明
eventMethod	文字	→ 発動されるイベントメソッド, または 空の文字の場合イベントの遮断を停止
processName	文字	→ プロセス名

説明

ON EVENT CALL コマンドは、イベントを検知するメソッドであるeventMethodをインストールします。このメソッドは、**イベント処理メソッド**または**イベントキャッチメソッド**と呼ばれます。

Tip: このコマンドの使用には、上級のプログラミング知識が必要です。通常、イベントを用いて作業を実行する際に、ON EVENT CALLコマンドを使用する必要はありません。フォームの使用において、イベントは4Dによって管理され、適切なフォームやオブジェクトにイベントが送信されます。

Tip: GET MOUSEやShift down等のコマンドを使用して、イベントに関する情報を取得できます。これらのコマンドをオブジェクトメソッドでコールし、オブジェクトに関連するイベントについての必要な情報を取得することができます。これらのコマンドにより、ON EVENT CALLコマンドをもとにしたアルゴリズムを作成する必要がなくなります。

このコマンドのスコープは、現在の作業セッションです。デフォルトで、別々のローカルプロセス内でメソッドが実行されます。いちどに1つのイベント処理メソッドだけを使用できます。メソッドを用いたイベントの検知を中止するには、eventMethodに空の文字列を指定して再度ON EVENT CALLコマンドをコールします。

イベント管理メソッドは別プロセスとして実行されるため、4Dメソッドが1つも実行されなくても、常にアクティブになります。インストール後はイベントが発生するたびに4Dがイベント処理メソッドを呼び出します。管理できるイベントは、マウスボタンのクリックとキーボードからの入力です。

オプションのprocessName引数は、ON EVENT CALLコマンドで作成されるプロセスの名前です。processNameの先頭にドル記号 ($) を付けるとローカルプロセスが開始されます。通常はこのローカルプロセスを使用します。processName引数を省略した場合、デフォルトで4Dは$Event Managerという名前のプロセスを作成します。

警告: イベント処理メソッドで実行する処理については十分注意してください。イベントを発生させるコマンドはコールしないでください。このようなコマンドをコールした場合、イベント処理メソッドの実行から抜けるのが非常に困難になります。Ctrl+Shift+Backspace（Windows）またはCommand-Shift-Option-Control-Backspace（Macintosh）キーにより、イベント管理プロセスを通常のプロセスに切り替えることができます。これによって、それ以降、発生するすべてのイベントは、イベント処理メソッドへ自動的には渡されなくなります。イベント管理が正常な動作ではなくなった場合に、この手法を利用して回復することが可能です。

イベント処理メソッドでは以下のシステム変数を読み取ることができます: MouseDown, KeyCode, Modifiers, MouseX, MouseY, そして MouseProc。これらの変数がプロセス変数であるという点に注意してください。したがって、変数のスコープはイベント処理プロセス内です。別のプロセスでこれらの値が必要な場合、インタープロセス変数へコピーしてください。

- システム変数MouseDownには、イベントがマウスクリックである場合には1が、それ以外の場合には0が代入されます。
- システム変数KeyCodeには、押されたキーのコードが代入されます。この変数は文字コードまたはファンクションキーコードを返します。これらのコードは、(さらにそのサブセクション)、およびの節にリストがあります。4Dは主要な文字コードとファンクションキーコードにたいして定義済みの定数があります。エクスプローラウィンドウでこれらの定数のテーマを参照してください。
- システム変数Modifiersには、モデファイアキーの値が入ります。システム変数Modifiersはイベントが発生した時点で、次のモディファイアキーが押されていたかどうかを示します:

プラットフォーム	**モディファイア**
Windows	Shift キー, Caps Lock, Alt キー, Ctrl キー, 右マウスボタン
Macintosh	Shift キー, Caps Lock, Option キー, Command キー, Control キー

Notes

- WindowsのALTキーは、MacintoshのOptionキーに相当します。
- WindowsのCtrlキーは、MacintoshのCommandキーに相当します。
- MacintoshのControlキーは、Windowsには対応するキーはありませんが、MaciontoshのControl＋マウスクリックは、Windowsのマウス右クリックに相当します。

モディファイアキーでは、イベントは発生しません。マウスボタンをクリック、あるいは他のキーが押されなければなりません。Modifiersは4バイトの倍長整数変数で、32ビットの配列としてとらえる必要があります。各モディファイアキーに対応するビットを判定するために、4Dにはビット位置やビットマスクを示す定数があらかじめ定義されています。例えば、イベントに対してshiftキーが押されたかどうかを調べるには、次のようにします:

```
If(Modifiers?? Shift key bit) ` Shiftキーが押されたか
```

または:

```
If((Modifiers & Shift key mask)#0) ` Shiftキーが押されたか
```

Note: Windowsでは、Modifiersとして128が使用でき、イベント生成時にマウスの左ボタンがリリースされたことを示します。

- システム変数MouseXとMouseYには、マウスがクリックされた時の座標位置が入ります。この位置は、クリックが発生したウインドウのローカル座標システムを用いて表現されます。ウインドウの左上隅の位置が0.0です。これらはマウスクリックした場合にのみ有効です。
- システム変数MouseProcには、イベントが発生した（マウスクリック）プロセスのプロセス参照番号が入ります。

重要: システム変数MouseDown, KeyCode, Modifiers, MouseX, MouseY, そして MouseProcには、ON EVENT CALLコマンドでインストールされたイベント処理メソッド内でのみ意味を持つ値が納められます

例題

次の例は、ユーザがCtrl+ピリオド（Windows）またはCommand+ピリオド（Macintosh）を押したら、印刷を中止します。まず、イベント処理メソッドをインストールします。次にユーザにメッセージを表示して、印刷を中止できることを知らせます。イベント処理メソッド内でインタープロセス変数<>vbWeStopにTrueが代入されると、既に印刷されたレコードの数をユーザに知らせます。最後にイベント処理メソッドをクリアします:

```
PAGE SETUP
If(OK=1)
   <>vbWeStop:=False
   ON EVENT CALL("EVENT HANDLER") ` イベント処理メソッドをインストール
   ALL RECORDS([People])
   MESSAGE("中断するにはCtrl+ピリオドを押す")
   $vlNbRecords:=Records in selection([People])
   For($vlRecord;1;$vlNbRecords)
      If(<>vbWeStop)
         ALERT("Printing cancelled at record "+String($vlRecord)+" of "+String($vlNbRecords))
         $vlRecord:=$vlNbRecords+1
      Else
         Print form([People];"Special Report")
      End if
   End for
   PAGE BREAK
   ON EVENT CALL("") ` イベント処理メソッドをアンインストール
End if
```

Ctrl+ピリオドが押されると、イベント処理メソッド内で<>vbWeStopがTrueに設定されます:

```
  ` EVENT HANDLER project method
If((Modifiers?? Command key bit) & (KeyCode=Period))
   CONFIRM("Are you sure?")
   If(OK=1)
      <>vbWeStop:=True
      FILTER EVENT ` この呼び出しを忘れないでください。そうでないと、4Dもこのイベントを受け取ります
   End if
End if
```

この例題では、ON EVENT CALLが使用されている点に注意してください。これは、ループでPAGE SETUP、PRINT FORM、PAGE BREAKコマンドを使い、特別な印刷用レポートを生成しているためです。

PRINT SELECTIONコマンドを使ってレポートを印刷する場合、ユーザに印刷を中断させるようなイベント処理を実行する必要はありません。この処理はPRINT SELECTIONコマンドにより自動的に行われます。

SET ASSERT ENABLED

SET ASSERT ENABLED (assertions {; *})

引数	型		説明
assertions	ブール	→	True = アサーションを有効にする False = アサーションを無効にする
*	演算子	→	省略時 = すべてのプロセスにコマンドを適用 (既存および後で作成されるものを含む) 指定時= カレントプロセスのみにコマンドを適用

説明

SET ASSERT ENABLED コマンドを使用してアプリケーションの4Dコードに挿入されたアサーションを無効にしたり、再度有効にしたりできます。アサーションに関する詳細はASSERTコマンドの説明を参照してください。

デフォルトで、プログラムに追加されたアサーションは有効です。このコマンドは、ときにアサーションの評価に実行時間のコストが必要であったり、アプリケーションのエンドユーザからアサーションを隠ぺいしたいなどの理由で、アサーションを無効にしたいときに使用できます。典型的には、SET ASSERT ENABLEDコマンドをで使用して、アプリケーションがテストモードや運用モードかによって、アサーションを有効にしたり無効にしたりします。

デフォルトでSET ASSERT ENABLEDコマンドはアプリケーションのすべてのプロセスに効果を及ぼします。コマンドの効果をカレントプロセスに限定するには、* 引数を渡します。

アサーションが無効にされていると、ASSERTコマンドに渡された式は評価されないことに留意してください。ASSERTを呼び出すコード行は動作においてもパフォーマンスにおいても、アプリケーションの処理に一切影響を及ぼしません。

例題

アサーションを無効にする:

```
SET ASSERT ENABLED(False)
ASSERT(TestMethod) // アサーションが無効なのでTestMethodは呼び出されない
```

印刷

- WindowsにおけるPDFCreatorドライバーの統合
- ACCUMULATE
- BREAK LEVEL
- CLOSE PRINTING JOB
- Get current printer
- Get print marker
- GET PRINT OPTION
- Get print preview
- GET PRINTABLE AREA
- GET PRINTABLE MARGIN
- Get printed height
- Is in print preview
- Level
- OPEN PRINTING JOB
- PAGE BREAK
- PAGE SETUP
- Print form
- PRINT LABEL
- Print object
- PRINT OPTION VALUES
- PRINT RECORD
- PRINT SELECTION
- PRINT SETTINGS
- PRINTERS LIST
- Printing page
- SET CURRENT PRINTER
- SET PRINT MARKER
- SET PRINT OPTION
- SET PRINT PREVIEW
- SET PRINTABLE MARGIN
- Subtotal

WindowsにおけるPDFCreatorドライバーの統合

4D v12でWindowsにおけるPDF印刷のサポートが拡張されました。PDF印刷を行うために、4DはPDFCreatorドライバに依存します。SET PRINT OPTIONとGET PRINT OPTIONはこのドライバを使用できるようにするために更新されました。

PDFCreatorはオープンソースの無料のドライバで、AFPL (Aladdin Free Public License) により管理されています。このライセンスについての詳細は以下のWebページを参照してください: http://en.pdfforge.org/content/license

PDFCreatorドライバを使用するためには、適切なバージョンのドライバをダウンロードしてご利用の環境にインストールする必要があります。4Dはデフォルトでインストールしません。4D v12が対応しているPDFCreatorドライバのバージョンは**0.9.9**以降です (バージョン**1.2**以上推奨)。このバージョンは以下のアドレスで入手できます: http://sourceforge.net/projects/pdfcreator/files/PDFCreator/PDFCreator/

ドライバをインストールするには管理者権限が必要です。

インストール時、デフォルトで"PDFCreator"という名前の新しい仮想プリンタがシステムにインストールされます。この名前は変更できます。

Note: Mac OSではシステムがネイティブにPDF印刷をサポートしています。

ACCUMULATE

ACCUMULATE (data {; data2 ; ... ; dataN})

引数	型		説明
data	フィールド, 変数	→	累計する数値型のフィールドまたは変数

説明

ACCUMULATEは、PRINT SELECTIONコマンドを使ってプリントするフォームレポート中で累計するフィールドまたは変数を指定します。

ブレーク処理を行うレポートを印刷する前に、BREAK LEVELコマンドとACCUMULATEコマンドを実行しなければなりません。これらのコマンドはレポートに対するブレーク処理を有効にします。Subtotalコマンドの説明を参照してください。

フォームレポートに数値フィールドまたは変数の小計を求める場合に、ACCUMULATEコマンドを使用します。ACCUMULATEは、4Dに対して、data毎の小計を記憶するように指示します。小計はBREAK LEVELコマンドで指定された各ブレークレベルに対して累計されます。

PRINT SELECTIONコマンドを使ってレポートを印刷する前に、ACCUMULATEコマンドを実行します。

フォームメソッドかオブジェクトメソッドでSubtotalコマンドを使用して、data引数の1つの小計を求めます。

例題

BREAK LEVEL コマンドの例題参照

BREAK LEVEL

BREAK LEVEL (level {; pageBreak})

引数	型	説明
level	倍長整数	⇒ブレークレベルの数
pageBreak	倍長整数	⇒改ページを行うブレークレベル

説明

BREAK LEVELは、PRINT SELECTIONコマンドを使ってプリントするレポートのブレークの数を指定します。

ブレーク処理を行うレポートを印刷する前に、**BREAK LEVEL**コマンドとACCUMULATEコマンドを実行しなければなりません。これらのコマンドはレポートに対するブレーク処理を有効にします。Subtotalコマンドの説明を参照してください。

level引数は、ブレーク処理を実行するもっとも深いレベルです。少なくとも同数のレベルでレコードをソートしなければなりません。ブレークレベルよりも多いレベルでソートすると、これらのレベルはソートされたものとして印刷されますが、ブレークに対しての意味は持ちません。

生成される各ブレークレベルは、フォーム中の対応するブレークエリアやヘッダエリアを印刷します。フォーム中のブレークエリアは、最低でもlevelの数だけ存在しなければなりません。フォーム中により多くのブレークエリアがある場合、それらは無視され、印刷されません。

2番目のオプションの引数pageBreakは、印刷中にページブレークを発生させるために使用します。

例題

以下の例は、2つのブレークレベルを持つレポートを印刷します。このセレクションは4つのレベルに対してソートされますが、**BREAK LEVEL**コマンドは2つのレベルだけにブレークを指定します。一つのフィールドはACCUMULATEコマンドで累計しています:

```
ORDER BY([Emp]Dept;>;[Emp]Title;>;[Emp]Last;>;[Emp]First;>) // 4つのレベルでソート
BREAK LEVEL(2) // 2つのレベルに対してブレーク処理を有効に(Dept とTitle)
ACCUMULATE([Emp]Salary) // 給与の累計
FORM SET OUTPUT([Emp];"Dept salary") // レポート用のフォームを選択
PRINT SELECTION([Emp]) // レポートを印刷する
```

CLOSE PRINTING JOB

CLOSE PRINTING JOB
このコマンドは引数を必要としません

説明

CLOSE PRINTING JOB コマンドは、OPEN PRINTING JOB コマンドで開かれたプリントジョブを閉じ、組み立てたプリントドキュメントをカレントプリンタに送信するために使用できます。

このコマンドが実行されると、プリンタは他のプリントジョブに対し利用可能となります。

Get current printer

Get current printer -> 戻り値

引数	型		説明
戻り値	文字	↩	カレントプリンタ名

説明

Get current printer コマンドは、4Dアプリケーションに定義されたカレントプリンタ名を返します。デフォルトで4Dの起動時には、システムで定義されたプリンタがカレントプリンタとなります。

プリントサーバ (スプーラ) を使用してカレントプリンタを管理している場合、完全なアクセスパス (Windows) またはスプーラの名前 (Mac OS) が返されます。

使用できるプリンター覧および追加情報を取得するにはPRINTERS LISTコマンドを使用します。カレントプリンタを変更するには、SET CURRENT PRINTERを使用します。

システム変数およびセット

プリンタがインストールされていなければOK変数は0に、そうでなければ1に設定されます。

Get print marker

Get print marker (markNum) -> 戻り値

引数	型		説明
markNum	倍長整数	→	マーカ番号
戻り値	倍長整数	↩	マーカの位置

説明

Get print marker コマンドを使用し、印刷中にマーカの現在位置を取得することができます。

このコマンドは次の2つの状況で使用することができます:

- PRINT SELECTIONおよびPRINT RECORDコマンドのコンテキストでのOn Headerフォームイベント中で。
- Print formコマンドのコンテキストでのOn Printing Detailフォームイベント中で。

座標はピクセル単位で返されます (1ピクセル＝1/72インチ)。

markNum引数には、テーマ内の定数のうちのいずれかを渡します:

定数	型	値
Form Break0	倍長整数	300
Form Break1	倍長整数	301
Form Break2	倍長整数	302
Form Break3	倍長整数	303
Form Break4	倍長整数	304
Form Break5	倍長整数	305
Form Break6	倍長整数	306
Form Break7	倍長整数	307
Form Break8	倍長整数	308
Form Break9	倍長整数	309
Form Detail	倍長整数	0
Form Footer	倍長整数	100
Form Header	倍長整数	200
Form Header1	倍長整数	201
Form Header10	倍長整数	210
Form Header2	倍長整数	202
Form Header3	倍長整数	203
Form Header4	倍長整数	204
Form Header5	倍長整数	205
Form Header6	倍長整数	206
Form Header7	倍長整数	207
Form Header8	倍長整数	208
Form Header9	倍長整数	209

例題

SET PRINT MARKERコマンドの例を参照

GET PRINT OPTION

GET PRINT OPTION (option ; value1 {; value2})

引数	型		説明
option	倍長整数, 文字	→	オプション番号
value1	倍長整数, 文字	←	オプションの値1
value2	倍長整数	←	オプションの値2

説明

GET PRINT OPTION コマンドは、プリントオプションの現在の値を返します。

option引数を使用して取得するオプションを指定することができます。""テーマ内にある定義済定数のいずれか (倍長整数)、またはPDFオプションコード (文字列) を渡すことができます:

オプション定数は以下の通りです:

定数	型	値
Paper option	倍長整数	1
Orientation option	倍長整数	2
Scale option	倍長整数	3
Number of copies option	倍長整数	4
Paper source option	倍長整数	5
Color option	倍長整数	8
Destination option	倍長整数	9
Double sided option	倍長整数	11
Spooler document name option	倍長整数	12
Mac spool file format option	倍長整数	13
Hide printing progress option	倍長整数	14

PDFオプションコードは2つの部分、*OptionType*と*OptionName*からなり、"*OptionType:OptionName*"のように組み合わせて使用します。

このコマンドは、指定したoptionの現在の値をvalue1と (オプションの) value2に返します。

オプション、PDFオプションコード、および考えられる値に関する詳細は、SET PRINT OPTIONコマンドの説明を参照してください。

次に示すGET PRINT OPTIONコマンド特定の機能に注意してください:

- option = 1 (paper option): value2を省略した場合、value1には現在の用紙名が返されます。value2を渡した場合、このコマンドは用紙の高さと幅をそれぞれvalue1とvalue2に返します。プリンタが提供するすべての用紙フォーマットの名前、高さ、幅を取得するには、PRINT OPTION VALUESコマンドを使用してください。
- option = 2 (orientation option): 1 (縦方向) または2 (横方向) が返されます。異なる方向オプションが使用されている場合、value1には0が代入されます。
- option = 5 (paper source option): value1には使用する用紙トレイの (PRINT OPTION VALUESコマンドから返されたトレイ配列の) インデックスが返されます (value2は必ず省略してください) 。
  Note: このオプションはWindows上でのみ使用できます。
- option = 8 (color option): value1にはカラー処理モードを表わすコードが返されます (1= 白黒 、2=カラー) 。
  Note: このオプションはWindows上でのみ使用できます。
- option = 9 (destination option): 定義済リスト上に現在の値が存在しない場合、value1には-1が代入され、システム変数OKには1がセットされます。エラーが発生した場合、value1とシステム変数OKには0が代入されます。value1に1または5以外の定義済の値が代入されている場合、value2には印刷されたファイルのアクセスパスが納められます。
- option = 11 (double sided option): value1に、0 (デフォルト値である標準か片面印刷) または1 (両面) が返されます。value1が1の場合、value2には0=左とじ (デフォルト) または1=上とじのいずれかの値が返されるかもしれません。
  Note: このオプションはWindows上でのみ使用できます。
- option = 12 (spooler document name option): 事前に定義されている場合、value1には現在の印刷文書の名前が返されます。事前定義されていない場合には、空の文字列が返されます。

Note: GET PRINT OPTION コマンドは、PostScriptプリンタに対してのみ動作します。

システム変数およびセット

このコマンドが正しく実行されると、OKシステム変数に1が、そうでなければ0が設定されます。

Get print preview

Get print preview -> 戻り値

引数	型	説明
戻り値	ブール	True: 印刷プレビューを行う False: 印刷プレビューを行わない

説明

Get print previewコマンドはカレントプロセスでSET PRINT PREVIEWが**True**で呼ばれている場合、Trueを返します。

ユーザーは印刷ダイアログでこのオプションを変更できる点に留意してください。最終的な印刷モードを判定するにはIs in print previewコマンドを使用します。

GET PRINTABLE AREA

GET PRINTABLE AREA (height {; width})

引数	型		説明
height	倍長整数	←	印刷可能領域の高さ
width	倍長整数	←	印刷可能領域の幅

説明

GET PRINTABLE AREA コマンドは、印刷可能領域のサイズを引数heightおよびwidthへピクセル単位で返します。このサイズは、現在の印刷設定、用紙方向等によって異なります。

返されるサイズは、各ページごとに変わることはありません（例えばページブレークの後）。

このコマンドをGet printed height コマンドと一緒に使用すると、印刷可能なピクセル数の認識やページ上のオブジェクトのセンタリングを行うことができるので便利です。

Note: 印刷管理ならびに4D用語に関する詳細は、GET PRINTABLE MARGINコマンドの説明を参照してください。

ページの全体のサイズを知るには:

- このコマンドで返された値にGET PRINTABLE MARGINコマンドで取得したマージンを加算します。
- または、次の構文を使用します:

```
SET PRINTABLE MARGIN(0;0;0;0) ` 用紙マージンを設定
GET PRINTABLE AREA(hPaper;wPaper) ` 用紙サイズ
```

GET PRINTABLE MARGIN

GET PRINTABLE MARGIN (left ; top ; right ; bottom)

引数	型		説明
left	倍長整数	⇐	左マージン
top	倍長整数	⇐	上マージン
right	倍長整数	⇐	右マージン
bottom	倍長整数	⇐	下マージン

説明

GET PRINTABLE MARGIN コマンドは、Print formコマンドに使用されるマージンの現在値を返します。

値は、用紙の端からの長さがピクセル単位で返されます。

GET PRINTABLE AREAコマンドを使用すると、用紙サイズの取得、および印刷可能領域の計算を行うことができます。

印刷可能なマージンの管理

デフォルトで、4Dにおける印刷に関する計算は"印刷可能マージン"に基づいています。このシステムの利点は、フォームが自動的に新しいプリンタに適応するということです（フォームは印刷可能領域に配置されるため）。その一方で、事前に印刷されたフォームの場合は、プリンタを変更すると印刷可能マージンが変わる可能性があるため、正確に印刷されるように各項目を配置することができませんでした。

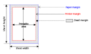
4Dバージョン6.8.1以降、Print form、PRINT RECORDおよびPRINT SELECTIONコマンドを使用して行うフォームの印刷を、各プリンタ共通の固定マージン（用紙マージン、つまり用紙の物理的な許容限界寸法）に基づいて行うことができるようになりました。これを行うには、GET PRINTABLE MARGIN、SET PRINTABLE MARGINおよびGET PRINTABLE AREAコマンドを使用します。

印刷用語について

用紙マージン: 用紙マージンは、用紙の物理的な許容限界寸法に相当します。

プリンタマージン: プリンタマージンは、その位置を越えてプリンタが印刷できないことを示すマージンです（プリントローラー、プリンタヘッドの行程終端等の物質的な理由により）。この値は、プリンタやフォーマットごとに異なります。

デッドマージン: 用紙マージンとプリンタマージンとの間の領域のことです。



Get printed height

Get printed height -> 戻り値

引数	型		説明
戻り値	倍長整数	↩	マーカーの位置

説明

Get printed height コマンドは、Print form コマンドを使って印刷された部分全体の高さ（ピクセル単位）を返します。

返される値は、0（ページ上端）からGET PRINTABLE AREAコマンドによって返される全体の高さ（印刷可能領域の最大サイズ）までの間の値です。

Print form コマンドを使って新しいセクションを印刷する場合は、新しいセクションの高さがこの値に加えられます。使用できる印刷可能領域がこのセクションの印刷に不十分である場合は、新しいページが作成され、値0が返されます。

右および左の印刷可能マージンは、上および下のマージン（SET PRINTABLE MARGINコマンドを使って指定可能）とは異なり、返される値に影響を与えません。

Note: 印刷管理ならびに4D用語に関する詳細は、GET PRINTABLE MARGINコマンドの説明を参照してください。

Is in print preview

Is in print preview -> 戻り値

引数	型	説明
戻り値	ブール	True: 印刷プレビュー False: 印刷プレビューでない

説明

Is in print previewコマンドは印刷ダイアログボックスで印刷プレビューオプションが選択されている場合Trueを返し、そうでなければFalseを返します。この設定はカレントプロセスに対し有効です。

Get print previewコマンドをと異なり、**Is in print preview**はユーザーがダイアログボックスでの設定を終えた後の、オプションの最終的な値を返します。なのでこのコマンドを使用すれば実際に印刷がプレビューモードで行われるのかどうかを判定できます。

例題

この例題はすべてのタイプの印刷に対応します:

```
SET PRINT PREVIEW(True) //デフォルトで印刷プレビューを行う
PRINT SETTINGS
If(OK=1)
 //ユーザーが印刷先を変更しているかもしれない
   If(Is in print preview) // 印刷プレビューならTrue
      FORM SET OUTPUT([Invoices];"toScreen")
   Else
      FORM SET OUTPUT([Invoices];"toPrinter"
   End if
   OPEN PRINTING JOB
   ALL RECORDS([Invoices])
   PRINT SELECTION([Invoices];>)
   CLOSE PRINTING JOB
End if
```

Level

Level -> 戻り値

引数	型	説明
戻り値	倍長整数	↩ カレントのブレークまはたヘッダのレベル

説明

Levelは、現在のブレークまたはヘッダのレベルを調べるために使用します。このコマンドは、On HeaderおよびOn Printing Breakイベント中でレベル数を返します。

レベル0は、印刷される最後のレベルで、総合計を印刷するのに適しています。Levelは、最初のソートフィールドのブレークを印刷するときに1を返し、2番目ソートフィールドでブレークを印刷するときに2を返します。

例題

以下の例はフォームメソッドのテンプレートです。集計レポートでフォームが出力フォームとして使用される際に、発生する可能性のあるイベントをすべて示しています。ヘッダやブレークがプリントされるときにLevelが呼び出されます:

```
 ` 集計レポートの出力フォームメソッド
$vpFormTable:=Current form table
Case of
 ` ...
   :(Form event=On Header)
 ` ヘッダエリアが印刷されようとしている
      Case of
         :(Before selection($vpFormTable->))
 ` 最初のヘッダブレークのコード
         :(Level=1)
 ` Level 1のヘッダブレークコード
         :(Level=2)
 ` Level 2のヘッダブレークコード
 ` ...
      End case
   :(Form event=On Printing Details)
 ` レコードが印刷されようとしている
 ` 各レコード毎のコード
   :(Form event=On Printing Break)
 ` ブレークエリアが印刷されようとしている
      Case of
         :(Level=0)
 ` ブレークLevel 0のコード
         :(Level=1)
 ` ブレークLevel 1のコード
 ` ...
      End case
   :(Form event=On Printing Footer)
      If(End selection($vpFormTable->))
 ` 最後のフッタのコード
      Else
 ` フッタのコード
      End if
End case
```

OPEN PRINTING JOB

OPEN PRINTING JOB
このコマンドは引数を必要としません

説明

OPEN PRINTING JOB コマンドはプリントジョブを開き、CLOSE PRINTING JOB コマンドが呼ばれるまで、続くすべてのプリント命令をスタックします。このコマンドはプリントジョブのコントロールを可能にし、特に印刷中に他のプリントジョブが予期せず挿入されないようにします。

OPEN PRINTING JOB コマンドはすべての4D印刷コマンド、クイックレポート、4D Writeや4D Viewプラグインの印刷コマンドで使用できます。他方このコマンドは4D Chartや4D Drawプラグイン、およびほとんどのサードパーティープラグインとは互換がありません。さらに、Windowsにおいてはプリントプレビューを行うことができません。

このコマンドでプリントジョブが開かれると、プリントが実際に起動されるまで、プリンタは"busy"モードに置かれます。このコンテキストで互換のないプラグインがプリントジョブを起動すると、"printer busy"エラーが返されます。

CLOSE PRINTING JOB コマンドを呼び出してプリントジョブを終了し、印刷ドキュメントをプリンタに送信しなければなりません。このコマンドを呼び出さないと、印刷ドキュメントはスタックに置かれたままとなり、4Dアプリケーションを終了するまでプリンタは利用可能になりません。

プリントジョブはプロセスに対しローカルですが、印刷はグローバルレベルで実行されます。4D内で一度にひとつだけプリントジョブを開くことができます。

OPEN PRINTING JOBはカレントの印刷設定を使用します (デフォルト設定またはPAGE SETUPやSET PRINT OPTIONコマンドで設定された設定)。印刷設定を変更するコマンドはOPEN PRINTING JOBが呼ばれる前に実行されなければなりません。そうでなければエラーが生成されます。

PAGE BREAK

PAGE BREAK {(* | >)}
* | > ➡*: Print formで開始されたプリントジョブをキャンセル, または >: 1つのプリントジョブを強制

説明

PAGE BREAKはプリンタに送信されたデータの印刷を実行させ、改ページを行います。PAGE BREAKは (On Printing Detailフォームイベントのコンテキストで) Print formと共に使用し、強制的に改ページを行ったり、メモリに作成された最後のページを印刷するために使用します。PAGE BREAKは、PRINT SELECTIONコマンドとともに使用してはいけません。この代りに、SubtotalやBREAK LEVELにオプション引数を使用してページブレークを行ってください。

* と > 引数は両方ともオプションです。

* 引数により、Print form コマンドによって開始したプリントジョブをキャンセルすることができます。このコマンドを実行すると、進行中のプリントジョブが直ちに中止されます。

Note: Windowsでは、プリンタサーバのスプールプロパティによってはこのメカニズムが動作しないことがあります。プリンタがただちに印刷を行うように設定されている場合、取消しは機能しないでしょう。PAGE BREAK(*) コマンドの操作を有効にするには、プリンタ設定で最後のページがスプールされてから印刷を開始する設定を選んでください。

> 引数は、PAGE BREAKの振る舞いを変更します。このシンタックスは2種類の効果を持ちます:

- PAGE BREAKコマンドが引数なしで再度実行されるまで、プリントジョブを開いたままにします。
- プリントジョブに優先権を与えます。プリントジョブが終了するまで、他のプリントは行われません。
  2番目のオプションは、スプールされるプリントジョブとともに使用すると、特に有効です。> 引数はプリントジョブが1つのファイルにスプールされることを保証します。これはプリント時間を短縮させます。

Note: スクリーンをプリントする際、ユーザがプリントプレビューダイアログボックスのキャンセルボタンをクリックした場合、PAGE BREAKコマンドはシステム変数OKに0を代入します。

例題 1

Print form コマンドの例題参照

例題 2

SET PRINT MARKER コマンドの例題参照

PAGE SETUP

PAGE SETUP ({aTable ;} form)

引数	型	説明
aTable	テーブル	⇒フォームの属するテーブル, または 省略した場合、デフォルトテーブル
form	文字	⇒ページ設定に使用するフォーム名

説明

PAGE SETUPは、プリンタの用紙設定を、formに格納された用紙設定に設定します。用紙設定はデザインモードでフォームが保存される時に、フォームとともに保存されます。

以下3つの条件のもとでは、印刷ダイアログボックスが表示されず、デフォルトの用紙設定でプリントが行われます:

- オプション引数 * を指定してPRINT SELECTIONコマンドを呼び出した場合
- オプション引数 * を指定してPRINT RECORDコマンドを呼び出した場合
- PRINT SETTINGSコマンドを呼び出さずに、一連のPRINT FORMを呼び出した場合

この場合、PAGE SETUPコマンドを使用すると用紙設定ダイアログボックスを表示せずに、デフォルト以外の用紙設定を使用することができます。

例題

[Design Stuff]テーブルには複数の (空の) フォームが作成されています。フォーム"PS100"には縮尺率が100%の用紙設定が割り当てられています。またフォーム"PS90"には縮尺率が90%の用紙設定が割り当てられています。以下のプロジェクトメソッドにより、プリントの度に用紙設定ダイアログボックス (表示はされません) で縮尺率を指定しなくても、さまざまな縮尺率でテーブルのセレクションをプリントすることができます:

```
  ` AUTOMATIC SCALED PRINTING プロジェクトメソッド
  ` AUTOMATIC SCALED PRINTING ( Pointer ; String {; Long } )
  ` AUTOMATIC SCALED PRINTING ( ->[table]; "Output form" {; Scaling } )
 If(Count parameters>=3)
    PAGE SETUP([Design Stuff];"PS"+String($3))
    If(Count parameters>=2)
       OUTPUT FORM($1->;$2)
    End if
 End if
 If(Count parameters>=1)
    PRINT SELECTION($1->;*)
 Else
    PRINT SELECTION(*)
 End if
```

このプロジェクトメソッドを作成した後は、以下のように使用します:

```
  ` カレントのインボイスを検索
 QUERY([Invoices];[Invoices]Paid=False)
  ` サマリレポートを90%縮尺で印刷
 AUTOMATIC SCALED PRINTING(->[Invoices];"Summary Report";90)
  ` サマリレポートを50%縮尺で印刷
 AUTOMATIC SCALED PRINTING(->[Invoices];"Detailed Report";50)
```

Print form

Print form ({aTable ;} form {; area1 {; area2}}) -> 戻り値

引数	型		説明
aTable	テーブル	→	フォームが属するテーブル, または 省略した場合は、デフォルトテーブル
form	文字	→	印刷するフォーム
area1	倍長整数	→	印刷マーカ、または開始エリア(area2が指定された場合)
area2	倍長整数	→	終了エリア(area1が指定された場合)
戻り値	倍長整数	↩	印刷されたエリアの高さ

説明

Print formは、aTableのフィールドや変数の現在の値を使用してformを印刷します。通常は、印刷処理を完全に制御する必要のある非常に複雑なレポートを印刷するために使用します。Print formはレコード処理、ブレーク処理、改ページ処理を全く行いません。これらの処理はすべて開発者が行います。Print formは固定されたサイズの枠のなかにフィールドや変数を印刷します。

Print formは、フォームの印刷後に改ページを行わないため、同じページに異なるフォームを容易に配置することができます。したがって、Print formは、異なるテーブルや異なるフォームを含む複雑な印刷処理に最適です。フォーム間で改ページを強制的に行うにはPAGE BREAKコマンドを使用してください。印刷可能領域を超える高さのフォームの印刷を次のページに持ち越すには、PAGE BREAKコマンドを使用する前にCANCELコマンドを呼び出してください。

3つの異なるシンタックスを使用できます:

詳細エリアの印刷

シンタックス:

```
height:=Print form(myTable;myForm)
```

この場合、Print formはフォームの詳細エリア (ヘッダマーカと詳細マーカの間のエリア) だけを印刷します。

フォームエリアの印刷

シンタックス:

```
height:=Print form(myTable;myForm;marker)
```

この場合コマンドはmarkerで示されるセクションを印刷します。以下のテーマの定数のうちの1つをmarker引数に渡します:

定数	型	値
Form Break0	倍長整数	300
Form Break1	倍長整数	301
Form Break2	倍長整数	302
Form Break3	倍長整数	303
Form Break4	倍長整数	304
Form Break5	倍長整数	305
Form Break6	倍長整数	306
Form Break7	倍長整数	307
Form Break8	倍長整数	308
Form Break9	倍長整数	309
Form Detail	倍長整数	0
Form Footer	倍長整数	100
Form Header	倍長整数	200
Form Header1	倍長整数	201
Form Header10	倍長整数	210
Form Header2	倍長整数	202
Form Header3	倍長整数	203
Form Header4	倍長整数	204
Form Header5	倍長整数	205
Form Header6	倍長整数	206
Form Header7	倍長整数	207
Form Header8	倍長整数	208
Form Header9	倍長整数	209

任意のエリア部分の印刷

シンタックス:

```
height:=Print form(myTable;myForm;areaStart;areaEnd)
```

この場合、コマンドはareaStartおよびareaEnd引数間に含まれる部分を印刷します。値はピクセル単位で入力しなければいけません。

Print formによって返される値は印刷可能範囲の高さを表します。この値は、Get printed height コマンドに自動的に考慮されます。

Print formを使用する場合、印刷ダイアログボックスは表示されません。レポートはデザインモードでフォームに割り当てられた用紙設定を使用しません。Print formを実行する前に用紙設定を指定する方法は2通りあります:

- PRINT SETTINGSコマンドを使用する。この場合、ユーザが設定を行います。
- PAGE SETUPコマンドを使用する。この場合、用紙設定はプログラムで指定します。

Print formは、印刷するページをそれぞれメモリ中に作成します。各ページはメモリ中のページがいっぱいになるか、PAGE BREAKコマンドを実行すると印刷されます。Print formの使用後、最後のページの印刷を確実に行うためには、PAGE BREAKコマンドで終了しなければなりません。そうでないと、最後のページはメモリ中に残り印刷されません。

警告: このコマンドが OPEN PRINTING JOB で開かれた印刷ジョブのコンテキストで呼び出された場合、PAGE BREAK を使用して最後のページを印刷しようとしてはいけません。なぜなら、最後のページはCLOSE PRINTING JOB コマンドによって自動的に印刷されるからです。この状況で PAGE BREAK を使用した場合、空のページが印刷されます。

このコマンドは外部エリアとオブジェクト(例えば 4D Write や 4D Viewエリアなど)を印刷します。エリアはコマンドの実行の際に毎回リセットされます。

警告: サブフォームは、Print form では印刷はされません。そのようなオブジェクトを含んだフォームを一つだけ印刷したい場合は、代わりに PRINT RECORD を使用して下さい。

Print form は、1回だけフォームメソッドのOn Printing Detailイベントを生成します。

4D Server: このコマンドは、ストアドプロシージャのフレームワークにおいて4D Server上で実行することができます。この状況では、次の制約があります:

- サーバマシン上ではダイアログボックスを一切表示しないでください (特定の必要性がある場合を除く)。
- プリンタ関連の問題が発生しても (用紙切れ、プリンタ接続切断等) 、エラーメッセージは生成されません。

例題 1

以下の例はPRINT SELECTIONコマンドをエミュレートします。しかし、レコードが小切手用かデポジット用であるかによって2種類のフォームの1つを使用します:

```
QUERY([Register]) ` レコードを選択
If(OK=1)
   ORDER BY([Register]) ` レコードをソート
   If(OK=1)
      PRINT SETTINGS ` 印刷設定ダイアログを表示
      If(OK=1)
         For($vlRecord;1;Records in selection([Register]))
            If([Register]Type ="Check")
               Print form([Register];"Check Out") ` 小切手用のフォーム
            Else
               Print form([Register];"Deposit Out") ` デポジット用のフォーム
            End if
            NEXT RECORD([Register])
         End for
         PAGE BREAK ` 最後のページを印刷
      End if
   End if
End if
```

例題 2

SET PRINT MARKER コマンドの例題参照

PRINT LABEL

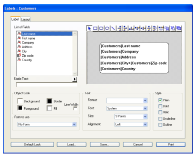
PRINT LABEL ({aTable }{;}{ document {; * | >}})
aTable　テーブル ⇒ 印刷するテーブル, または 省略した場合、デフォルトテーブル
document 文字 ⇒ ディスクに保存したラベルドキュメント名
* | > ⇒ *: 印刷ダイアログを省略, または >: 印刷設定の再初期化をしない

説明

PRINT LABELは、aTableのセレクションのデータを使用してラベルを印刷します。

document引数を指定しないと、PRINT LABELはカレント出力フォームを使用して、aTableのカレントセレクションをラベルとして印刷します。サブフォームの印刷にこのコマンドを使用することはできません。ラベルのフォーム作成に関する詳細は*4D Design Reference*マニュアルを参照してください。

document 引数を指定すると、PRINT LABELコマンドは、ラベルウィザード（下図）を表示するか、またはディスクに保存された既存のラベルドキュメントを印刷します。以下の説明を参照してください。


Labels : Customers
Label
Layout
List of Fields
Last name
First name
Company
Address
City
Zip code
Country
[Customers]Last name
[Customers]Company
[Customers]Address
[Customers]City+[Customers]Zip code
[Customers]Country
Static Text:
Object Look
Background
Border
Foreground
Fill
Line Width
Form to use
No Form
Text
Format
Font
System
Size
9 Points
Alignment
Left
Style
☑ Plain
☐ Bold
☐ Italic
☐ Underline
☐ Outline
Default Look
Load...
Save...
Cancel
Print

デフォルトで、PRINT LABELは印刷の前にプリンタダイアログボックスを表示します。ユーザが印刷ダイアログボックスのいずれかを取消すと、コマンドはキャンセルされラベルは印刷されません。
オプションの * または > 引数を使用することで、印刷ダイアログボックスを抑制することが出来ます:

- * 引数は、現在の印刷設定 (デフォルトの設定、もしくはPAGE SETUPやSET PRINT OPTIONコマンドで定義した) に従い、印刷処理を行います。
- > 引数は上記に加え、現在の印刷設定を再初期化することなく印刷を行います。この設定は、以前に設定した内容を継続し、(例えばループ中で) 連続したPRINT LABELコマンドを使用する時に役立ちます。この引数の使用例はPRINT RECORDコマンドの例を参照してください。

これらの引数はラベルウィザード使用時には効果を持たないことに注意してください。

ラベルウィザードを使用しない場合、ラベルがすべてプリントされるとシステム変数OKに1が代入されます。それ以外の場合には0が代入されます (印刷ダイアログボックスでキャンセルがクリックされた場合等)。

document 引数を指定すると、ラベルはdocumentに定義されたラベル設定情報に従って印刷されます。document に空の文字列 ("") を指定すると、PRINT LABELはファイルを開くダイアログボックスが表示し、ユーザはラベル設定として使用するファイルを指定することができます。documentに存在しないドキュメント名を指定すると (例えばchar(1)をdocumentに渡す) 、ラベルウィザードが表示され、ユーザはラベル設定を定義することができます。

Note: デザインモードで、tableが非表示に設定されている場合、ラベルウィザードは表示されません。

4D Server: このコマンドは、ストアドプロシージャのフレームワークにおいて4D Server上で実行することができます。この状況では、次の制約があります:

- サーバマシン上ではダイアログボックスを一切表示しないでください (特定の必要性がある場合を除く)。そのためにはこのコマンドを * または > 引数付きで呼び出さなければなりません。
- ラベルエディタが表示されるシンタックスは4D Serverでは動作しません。この場合システム変数OKは0に設定されます。
- プリンタ関連の問題が発生しても (用紙切れ、プリンタ接続切断等) 、エラーメッセージは生成されません。

例題 1

以下の例は、テーブルの出力フォームを使用してラベルを印刷します。この例では2つのメソッドを使用します。最初のプロジェクトメソッドは正しい出力フォームを設定し、ラベルを印刷します:

```
ALL RECORDS([Addresses]) ` 全レコードを選択
FORM SET OUTPUT([Addresses];"Label Out") ` 出力フォームを選択
PRINT LABEL([Addresses]) ` ラベルを印刷
FORM SET OUTPUT([Addresses];"Output") ` デフォルトの出力フォームに戻す
```

2つ目のメソッドはフォーム"**Label Out**"のフォームメソッドです。このフォームは、各フィールドの内容を連結した結果を格納するための1つの変数*vLabel*を含みます。フィールドAddr2が空の場合、メソッドにより取り除かれます。ラベルウィザードを使用しても、この処理が自動的に実行される点に注意してください。フォームメソッドは、各レコードに対して1つずつラベルを作成します:

```
 ` [Addresses]; "Label Out" フォームメソッド
Case of
   :(Form event=On Load)
      vLabel:=[Addresses]Name1+" "+[Addresses]Name2+Char(13)+[Addresses]Addr1+Char(13)
      If([Addresses]Addr2 #"")
         vLabel:=vLabel+[Addresses]Addr2+Char(13)
      End if
      vLabel:=vLabel+[Addresses]City+", "+[Addresses]State+" "+[Addresses]ZipCode
End case
```

例題 2

以下の例では、ユーザが[People]テーブルを検索し、自動で"My Labels"ラベルを印刷します:

```
QUERY([People])
If(OK=1)
   PRINT LABEL([People];"My Labels";*)
End if
```

例題 3

以下の例では、ユーザが[People]テーブルを検索し、印刷するラベルを選択します:

```
QUERY([People])
If(OK=1)
   PRINT LABEL([People];"")
End if
```

例題 4

以下の例では、ユーザが[People]テーブルを検索し、ラベルウィザードを表示して任意のラベルの設計、保存、ロード、印刷を行います:

```
QUERY([People])
If(OK=1)
   PRINT LABEL([People];Char(1))
End if
```

Print object

Print object ({* ;} object {; posX {; posY {; width {; height}}}}) -> 戻り値

引数	型		説明
*	演算子	→	指定時objectはオブジェクト名 (文字列) 省略時 objectは変数
object	フォームオブジェクト	→	オブジェクト名 (* 指定時) または 変数 (* 省略時)
posX	倍長整数	→	オブジェクトの横位置
posY	倍長整数	→	オブジェクトの縦位置
width	倍長整数	→	オブジェクトの幅 (ピクセル)
height	倍長整数	→	オブジェクトの高さ (ピクセル)
戻り値	ブール	↩	True = オブジェクトが完全に印刷された; そうでなければFalse

説明

Print object コマンドを使用して objectと* 引数で指定したフォームオブジェクトを、posXとposYの位置に、任意のサイズで印刷できます。

Print objectコマンドはプロジェクトフォーム上のオブジェクトの印刷のみに使用できます。このコマンド呼び出し前に、印刷するオブジェクトを含むプロジェクトフォームを新しいOPEN PRINTING FORMで指定しなければなりません。

オプションの* 引数を渡すと、object引数にはオブジェクト名 (文字列) を渡します。* 引数を渡さない場合、objectには変数を指定します。この場合、文字列ではなく変数参照 (オブジェクトタイプのみ) を渡します。

posXとposY引数はオブジェクトを印刷する開始位置を指定します。これらの値はピクセル単位で指定ます。これらの引数を省略すると、オブジェクトはフォーム上の位置に基づいて印刷されます。

widthとheight引数はフォームオブジェクトの幅と高さを指定します。Print objectコマンドは可変長のオブジェクトを管理しません。OBJECT GET BEST SIZEコマンドでオブジェクトのサイズを管理しなければなりません。またOBJECT GET BEST SIZEコマンドでテキストを含むオブジェクトのもっとも適切なサイズを知ることができます。同様に、Print objectはページブレークを自動では生成しません。必要に応じて開発者が管理しなければなりません。

4Dコマンドを使用してオブジェクトプロパティ (色やサイズなど) をオンザフライで変更できます。

オブジェクトが完全に印刷されるとコマンドはTrueを返します。そうでないばあい、言い換えればフレームワーク内のオブジェクトに割り当てられたデータをすべて印刷できなかった場合、コマンドはFalseを返します。特にリストボックスのすべての行を印刷できなかった場合、コマンドはFalseを返します。この場合Print objectコマンドを、それがTrueを返すまで繰り返し呼び出します。特別なメカニズムが自動で使用され、オブジェクトの内容が呼び出しごとに自動でスクロールされます。

注:

- 4D の現在のバージョンでは、リストボックスタイプのオブジェクトのみがこのメカニズムを持っています (他のオブジェクトではコマンドは常に Trueを返します)。4Dの将来のバージョンでこの機能は他の可変長オブジェクトに拡張されます。
- LISTBOX GET PRINT INFORMATIONコマンドを使用して、処理中に印刷状況を知ることができます。

Print objectコマンドは事前にOPEN PRINTING JOBで開かれた印刷ジョブのコンテキストでのみ使用できます。このコンテキストの外で呼び出された場合、コマンドはなにも行いません。同じ印刷ジョブ内でPrint objectコマンドを複数回呼び出すことができます。

注: 階層リスト、サブフォーム、およびWebエリアを印刷することはできません。

例題 1

フォーム上の10個のオブジェクトを印刷する例:

```
PRINT SETTINGS
If(OK=1)
   OPEN PRINTING JOB
   If(OK=1)
      FORM LOAD("PrintForm")
      x:=100
      y:=50
      GET PRINTABLE AREA(hpaper;wpaper)
      For($i;1;10)
         OBJECT GET BEST SIZE(*;"Obj"+String($i);bestwidth;bestheight)
         $end:=Print object(*;"Obj"+String($i))
         y:=y+bestheight+15
         If(y>hpaper)
            PAGE BREAK(>)
            y:=50
         End if
      End for
   End if
   CLOSE PRINTING JOB
End if
```

例題 2

リストボックスを完全に印刷する例:

```
Repeat
   $end:=Print object(*;"mylistbox")
Until($end)
```

PRINT OPTION VALUES

PRINT OPTION VALUES (option ; namesArray {; info1Array {; info2Array}})

引数	型		説明
option	倍長整数	⇨	オプション番号
namesArray	テキスト配列	⇦	値の名前
info1Array	倍長整数配列	⇦	オプションの値(1)
info2Array	倍長整数配列	⇦	オプションの値(2)

説明

PRINT OPTION VALUES コマンドは、optionで指定したプリントオプションに対して使用できる値の名称リストをnamesArray引数に返します。オプションとして、info1Arrayとinfo2Array内に各値に関する情報を取得することができます。

option引数に取得するオプションを指定します。必ず˝テーマ内にある次の定数のいずれかを渡してください (値名の一覧を返すことができるオプション):

定数	型	値
Paper option	倍長整数	1
Paper source option	倍長整数	5

コマンドの実行後、namesArray配列ならびに配列info1Arrayとinfo2Array (適用可能な場合) には、使用できる値の名称と情報が代入されます。

option 引数に値1 (paper option) を渡した場合、コマンドから次の情報が返されます:

- namesArrayには、使用できる用紙フォーマットの名前。
- info1Arrayには、各用紙フォーマットの高さ。
- info2Arrayには、各用紙フォーマットの幅。

Note: この情報を取得するには、プリンタドライバがそのプリンタの有効なPPD (PostScript Printer Description) へアクセスできなくてはなりません。

SET PRINT OPTIONコマンドを使用して特定の用紙フォーマットを適用するために、namesArrayのいずれかの値を渡すか、またはinfo1Arrayとinfo2Arrayの対応する値を渡すことができます。

option引数に値5 (paper source option) を渡した場合、このコマンドからは利用可能なトレイ名がnamesArrayに返され、info1Arrayには内部的なWindowsのID番号が返されます (info2Array は空のままです) 。
配列内の値の順序は、プリンタドライバにより決まります。SET PRINT OPTIONコマンドを使用してトレイを指定するには、namesArrayまたはinfo1Arrayから得ることのできる、配列要素のインデックスを渡さなければなりません。

Note: このオプションは、Windows上でのみ使用できます。

各種プリントオプションに関する詳細は、SET PRINT OPTIONとGET PRINT OPTIONコマンドの説明を参照してください。

これらのコマンドから返される情報はすべて、OSにより提供されます。特定のオプションについての詳細は、お使いのシステムのドキュメントを参照してください。

Note: PRINT OPTION VALUES コマンドコマンドは、ポストスクリプトプリンタに対してのみ動作します。

PRINT RECORD

PRINT RECORD ({aTable}{;}{* | >})

引数	型		説明
aTable	テーブル	→	カレントレコードを印刷するテーブル または 省略した場合はデフォルトテーブル
* | >	演算子	→	*: 印刷ダイアログを省略, または >: 印刷設定の再初期化をしない

説明

PRINT RECORD はtableのカレントレコードを、カレントセレクションを変更せずに印刷します。カレント出力フォームが印刷に用いられます。tableにカレントレコードが存在しない場合、PRINT RECORDは何も行いません。

PRINT RECORDコマンドを使ってサブフォームを印刷することができます。これはPrint formでは実行できません。

Note: レコードに対して行われた修正が保存されていない場合、ディスク上の修正前のフィールド値ではなく、修正後の値が印刷されます。

デフォルトでPRINT RECORDは印刷の前に印刷ダイアログを表示します。印刷ダイアログをユーザーがキャンセルした場合、PRINT RECORDはキャンセルされ、印刷は行われません。
このダイアログの表示を省略するには、オプション引数 * または > を使います:

- * 引数は、現在の印刷設定 (デフォルトの設定、もしくはPAGE SETUPやSET PRINT OPTIONコマンドで定義した) に従い、印刷処理を行います。
- > 引数は上記に加え、現在の印刷設定を再初期化することなく印刷を行います。この設定は、以前に設定した内容を継続し、(例えばループ中で) 連続したPRINT RECORDコマンドを使用する時に役立ちます。

4D Server: このコマンドは、ストアドプロシージャのフレームワークにおいて4D Server上で実行することができます。この状況では、次の制約があります:

- サーバマシン上ではダイアログボックスを一切表示しないでください (特定の必要性がある場合を除く)。そのためにはこのコマンドを * または > 引数付きで呼び出さなければなりません。
- プリンタ関連の問題が発生しても (用紙切れ、プリンタ接続切断等) 、エラーメッセージは生成されません。

警告: PRINT RECORDと一緒にPAGE BREAKコマンドを使用してはいけません。PAGE BREAKコマンドはPrint formと一緒に使用します。

例題 1

以下の例では、[Invoices] テーブルのカレントレコードを印刷します。このコードは入力フォームの**印刷**ボタンのオブジェクトメソッド内に記述されています。ユーザがそのボタンをクリックすると、レコードは指定した出力フォームで印刷されます。

```
FORM SET OUTPUT([Invoices];"Print One From Data Entry") ` 印刷用に作成された出力フォームを選択
PRINT RECORD([Invoices];*) ` 請求書をそのまま印刷 (印刷設定ダイアログを表示せずに)
FORM SET OUTPUT([Invoices];"Standard Output") ` 出力フォームを元に戻す
```

例題 2

以下の例では、同じカレントレコードを二種類の異なるフォームを使用して印刷しています。このコードは、入力フォームの**印刷**ボタンのオブジェクトメソッド内に記述されています。印刷設定をを行った後、二種類のフォームでその設定を使用することが出来ます。

```
PRINT SETTINGS `ユーザが印刷設定を行う
If(OK=1)
   FORM SET OUTPUT([Employees];"Detailed") `1番目の印刷フォーム
   PRINT RECORD([Employees];>) `ユーザが指定した設定を使用して印刷
   FORM SET OUTPUT([Employees];"Simple") `2番目の印刷フォーム
   PRINT RECORD([Employees];>) `ユーザが指定した設定を使用して印刷
   FORM SET OUTPUT([Employees];"Output") `出力フォームを元に戻す
End if
```

PRINT SELECTION

PRINT SELECTION ({aTable}{;}{* | >})

引数	型		説明
aTable	テーブル	⇒	セレクションを印刷するテーブル, または 省略した場合、デフォルトテーブル
* | >	演算子	⇒	*: 印刷ダイアログを省略, または >: 印刷設定の再初期化をしない

説明

PRINT SELECTIONは、aTableのカレントセレクションを印刷します。レコードは、カレントプロセスのテーブルのカレント出力フォームを使用して印刷されます。PRINT SELECTIONは、デザインモードのプリント...メニューと同じ動作を実行します。セレクションが空の場合、PRINT SELECTIONは何も行いません。

デフォルトで、PRINT SELECTIONは印刷前に印刷ダイアログボックスを表示します。ユーザが印刷ダイアログボックスでキャンセルを行った場合、コマンドはキャンセルされ、印刷を行いません。
オプション引数の * または > を使用して、ダイアログボックスの表示を取り消すことができます:

- * 引数は、現在の印刷設定 (デフォルトの設定、もしくはPAGE SETUPやSET PRINT OPTIONコマンドで定義した) に従い、印刷処理を行います。
- > 引数は上記に加え、現在の印刷設定を再初期化することなく印刷を行います。この設定は、以前に設定した内容を継続し、(例えばループ中で) 連続したPRINT SELECTIONコマンドを使用する時に役立ちます。この引数の使用例はPRINT RECORD コマンドの説明を参照してください。

印刷中には、デザインモードのプロパティリストウィンドウで有効にされたフォームおよびオブジェクトのイベントと、実際に発生しているイベントに応じて、出力フォームのフォームメソッドとオブジェクトメソッドが実行されます:

- On Headerイベントはヘッダを印刷する直前に生成されます。
- On Printing Detailイベントはレコードを印刷する直前に生成されます。
- On Printing Breakイベントはブレークエリアを印刷する直前に生成されます。
- On Printing Footerイベントははフッタを印刷する直前に生成されます。

PRINT SELECTIONが最初のヘッダを印刷しているかどうかは、On HeaderイベントでBefore selectionを判定することによって調べることができます。またOn Printing FooterイベントでEnd selectionを判定することによって、最後のフッタかどうかをチェックすることができます。これら関数の詳細は、それぞれのコマンドやForm event、Levelの説明を参照してください。

PRINT SELECTIONを使用し、小計やブレーク付きでセレクションを印刷するには、まずそのセレクションをソートしなければなりません。次に、レポートの各ブレークエリアに、小計を変数に代入するオブジェクトメソッドを持つ変数を配置します。変数に値を代入する、SumやAverageのような統計関数と算術関数を使用することもできます。詳細はSubtotal、BREAL LEVEL、ACCUMULATEコマンドの説明を参照してください。

警告: PRINT SELECTIONのコンテキストでPAGE BREAKコマンドを使用してはいけません。PAGE BREAKはPrint formのコンテキストで使用します。

PRINT SELECTIONの呼び出し後、プリントが正常に終了するとシステム変数OKに1がセットされます。プリントが中断された場合には、システム変数OKには0がセットされます（例えばユーザが印刷ダイアログボックスでキャンセルをクリックした場合）。

4D Server: このコマンドは、ストアドプロシージャのフレームワークにおいて4D Server上で実行することができます。この状況では、次の制約があります:

- サーバマシン上ではダイアログボックスを一切表示しないでください (特定の必要性がある場合を除く)。そのためにはこのコマンドを * または > 引数付きで呼び出さなければなりません。
- プリンタ関連の問題が発生しても (用紙切れ、プリンタ接続切断等) 、エラーメッセージは生成されません。

例題

以下の例は、最初に[People]テーブルのすべてのレコードを選択します。次にDISPLAY SELECTIONコマンドを使用してすべてのレコードを表示し、ユーザがプリントするレコードを選択します。最後にUSE SETコマンドにより、選択されたレコードをPRINT SELECTIONで印刷します:

```
ALL RECORDS([People]) ` 全レコード選択
DISPLAY SELECTION([People];*) ` レコード表示
USE SET("UserSet") ` ユーザが指定したレコードのみを使用
PRINT SELECTION([People]) ` 印刷実行
```

PRINT SETTINGS

PRINT SETTINGS {(dialType)}

引数	型		説明
dialType	倍長整数	→	表示するダイアログボックス: 0 (または省略時) = すべて 1 = 用紙設定, 2 = 印刷設定

説明

PRINT SETTINGSは印刷設定ダイアログボックスを表示します。このコマンドを使用する場合、Print formやOPEN PRINTING JOBコマンドよりも前に呼び出されなければなりません。

オプションのdialType引数を使用して、表示する印刷設定ダイアログボックスを指定できます:

- dialTypeに0を渡すか省略すると、両方のダイアログが表示されます。まず用紙設定ダイアログが表示され、次にプリントジョブダイアログが表示されます。
- dialTypeに1を渡すと、用紙設定ダイアログのみが表示されます。プリントジョブ設定はカレントの設定が使用されます。
- dialTypeに2を渡すと、プリントジョブダイアログのみが表示されます。用紙設定はカレントの設定が使用されます。

Note: プリントジョブダイアログボックスには、プレビューチェックボックスがあります。このチェックボックスでレポートを画面に印刷するように指定することができます。PRINT SETTINGSを実行する前に、SET PRINT PREVIEWコマンドを使用してこのチェックボックスをあらかじめセットする、またはセットし直すことができます。

例題

PRINT FORMコマンドの例題参照

システム変数およびセット

ユーザが両方のダイアログボックスでOKボタンをクリックすると、システム変数OKに1が代入されます。それ以外の場合は、システム変数OKに0が代入されます。

PRINTERS LIST

PRINTERS LIST (namesArray {; altNamesArray {; modelsArray}})

引数	型		説明
namesArray	テキスト配列	←	プリンタ名
altNamesArray	テキスト配列	←	・・・Windows: プリンタの場所 Mac OS: カスタムプリンタ名
modelsArray	テキスト配列	←	プリンタモデル (Windowsのみ)

説明

PRINTERS LIST コマンドは、引数として渡された各配列にそのマシンで使用できるプリンタの名前、およびオプションとしてプリンタの場所とモデルを返します。

Note: プリントサーバ (スプーラ) を使用してプリンタを管理している場合、フルアクセスパス (Windows) またはスプーラの名前 (Mac OS) が返されます。

namesArray引数には、テキスト配列を渡します。コマンドの実行後、この配列には使用できるプリンタの名前が代入されます。Mac OSの場合、固定のシステムの名称になります。

オプションで2番目の引数としてaltNamesArrayを渡すことができます。この配列に含まれる内容はプラットフォームにより異なります:

- Windowsでは、各プリンタに関して、ネットワーク・ロケーション(または、ローカル・ポート) が代入されます。
- Mac OSでは、各プリンタに関して、ユーザが変更することのできるカスタム名称が代入されます。例えば、ダイアログボックスの中でこの名前を使用することができます。

オプション引数modelsArrayを渡した場合、各プリンタのモデルを取得できます。この引数は、Windowsでのみ指定できます。

4Dで選択されたプリンタの変更や取得を行うには、SET CURRENT PRINTERとGet current printerコマンドを使用します。最初の配列 (namesArray) に返された名前を渡さなければなりません。

Windows上では、プリンタ名はOSレベルで手動にて変更することができます。一方、プリンタの場所とモデルタイプは、その物理的特性に関連しています。したがって、オプションの配列に返された値を使用して、選択したプリンタの特性を調べることができます。特に、クライアントマシンがすべて同じプリンタを使用していることをチェックすることができます。

Mac OS上では、プリンタ名 (プリンタサーバの名前) を使用して、このチェックを行います。このプリンタ名は、接続している各マシンに対して同じ名前になります。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定され、空の配列が返されます。

Printing page

Printing page -> 戻り値

引数	型		説明
戻り値	倍長整数	➲	現在印刷中のページのページ番号

説明

Printing pageは、印刷中のページ番号を返します。このコマンドは、PRINT SELECTIONコマンドまたはデザインモードのプリント...メニューの選択によって印刷する場合にのみ使用することができます。

例題

以下の例は、両面印刷フォーマットのレポートにページ番号を設定します。ページ番号の位置を変更するために、フォームはページ番号を表示する変数を2つ持っています。変数 (vLeftPageNum) は、偶数のページ番号を印刷します。変数 (vRightPageNum) は、奇数のページ番号を印刷します。この例は、偶数ページを判定し適切な変数に値を代入します:

```
Case of
   :(Form event=On Printing Footer)
      If((Printing page% 2)=0) ` モジューロが0, 偶数ページ
         vLeftPageNum:=String(Printing page) ` 左ページ番号を設定
         vRightPageNum:="" ` 右ページ番号をクリア
      Else ` 奇数ページ
         vLeftPageNum:="" ` 左ページ番号をクリア
         vRightPageNum:=String(Printing page) ` 右ページ番号を設定
      End if
End case
```

SET CURRENT PRINTER

SET CURRENT PRINTER (printerName)

引数	型		説明
printerName	文字	➡	使用されるプリンタ名

説明

SET CURRENT PRINTER コマンドは現行の4Dアプリケーションでの印刷に使用するプリンタを指定するために使用します。

printerName引数に選択するプリンタの名前を渡します。使用できるプリンター覧を取得するには、このコマンドの前にPRINTERS LISTコマンドを使用します。
printerNameに空の文字列を渡すと、システムに定義されたカレントプリンタが使用されます。

SET CURRENT PRINTERコマンドで、印刷先として、PDFCreatorドライバーがインストールした仮想プリンターを指定できます。4DはWindows上でPDFドキュメントの印刷を容易にするために、PDFCreatorドライバーに依存します (WindowsにおけるPDFCreatorドライバーの統合参照)。
PDFドキュメントを印刷するには、printerName引数にPDFCreatorドライバーがインストールした仮想プリンターの名前を渡します。仮想プリンターの名前はデフォルトで"PDFCreator"です。この名前はドライバーのインストール時に変更できます。4Dが自動で仮想プリンターの名前を検索して使用できるように、printerNameに定数PDFCreator Printer nameを渡してください。この定数はPrint Optionsテーマにあります。

SET CURRENT PRINTERコマンドは、利用可能なオプションが選択したプリンタに対応するように、必ずSET PRINT OPTIONコマンドよりも前に呼び出してください。それに対し、SET CURRENT PRINTERコマンドは、PAGE SETUPコマンドの後で呼び出さなければなりません。これを行わないと、印刷設定が失われます。

このコマンドはPRINT SELECTION、PRINT RECORD、Print form、およびQR REPORTコマンドと一緒に使用することができ、デザインモードを含め、4Dにおけるすべての印刷に対して適用されます。
指定した設定が失われないようにするには、必要に応じて印刷コマンドは必ず引数 > を用いて呼び出さなければなりません。

システム変数およびセット

プリンターの選択が正しく実行されると、システム変数OKに1が設定されます。そうでない場合 (例えば指定したプリンターが見つからない場合など)は、システム変数OKは0に設定され、カレントプリンターは変更されません。

SET PRINT MARKER

SET PRINT MARKER (markNum ; position {; *})

引数	型		説明
markNum	倍長整数	➡	マーカ番号
position	倍長整数	➡	マーカの新しい位置
*	演算子	➡	指定時 = 後続のマーカを移動する 省略時 = 後続のマーカを移動しない

説明

SET PRINT MARKER コマンドを使用し、印刷時にマーカ位置を指定することができます。このコマンドをGet print marker、OBJECT MOVE、Print formコマンドと組み合わせて使用することにより、印刷エリアのサイズを調節することができます。

SET PRINT MARKERは次の2つの状況において使用可能です:

- PRINT SELECTIONおよびPRINT RECORDコマンドのコンテキストでのOn headerフォームイベント中。
- Print formコマンドのコンテキストでのOn Printing Detailフォームイベント中。この処理はカスタマイズされたレポートの印刷を容易にします (例を参照)。

コマンドの有効範囲は印刷に限定され、画面上には変更が表示されません。フォームに対して行った変更は保存されません。

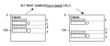
markNum引数には、テーマ内の定数のいずれかを渡します:

定数	型	値
Form Break0	倍長整数	300
Form Break1	倍長整数	301
Form Break2	倍長整数	302
Form Break3	倍長整数	303
Form Break4	倍長整数	304
Form Break5	倍長整数	305
Form Break6	倍長整数	306
Form Break7	倍長整数	307
Form Break8	倍長整数	308
Form Break9	倍長整数	309
Form Detail	倍長整数	0
Form Footer	倍長整数	100
Form Header	倍長整数	200
Form Header1	倍長整数	201
Form Header10	倍長整数	210
Form Header2	倍長整数	202
Form Header3	倍長整数	203
Form Header4	倍長整数	204
Form Header5	倍長整数	205
Form Header6	倍長整数	206
Form Header7	倍長整数	207
Form Header8	倍長整数	208
Form Header9	倍長整数	209

positionには、新しい位置をピクセル単位で渡します。

オプション引数 * を渡すと、このコマンドの実行時に、markNumで指定したマーカより下側に位置するすべてのマーカが、指定したマーカと同じピクセル数だけ、同じ方向へ移動します。

警告: このマーカより下側にあるエリア内のオブジェクトもすべて移動します。
引数 * を使用すると、後続の各マーカの最初の位置より下側にmarkNumで指定したマーカを配置することができます。これら後続のマーカも同時に移動します。

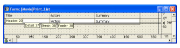


Notes:

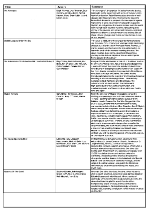
- このコマンドで変更できるのは、既存のマーカ位置だけです。マーカを追加することはできません。フォームに存在しないマーカを指定した場合、コマンドは何も行いません。
- デザインモードにおける印刷用マーカの仕組みは変わりません。つまり、あるマーカは、それより上にあるマーカを飛び越えて移動したり、後続のマーカよりも下側に移動することはできません(引数 * を使用しない場合)。

例題

この例は、3つのカラムがあるレポートを作成します。各行の高さは、フィールド内容に応じて実行中に計算されます。
印刷に使用する出力フォームは次の通りです:



このフォームに対してOn Printing Detailフォームイベントが選択されています (印刷されるエリアに関係なく、Print form コマンドはこのタイプのフォームイベントだけを生成する点に留意してください)。
レコードごとに、(多くの内容を保持する) "Actors"または"Summary"カラムの内容に応じて行の高さを調整しなくてはなりません。目的とする結果を次に示します:



印刷用のプロジェクトメソッドは次の通りです:

```
C_LONGINT(vLprint_height;$vLheight;vLprinted_height)
C_STRING(31;vSprint_area)
PAGE SETUP([Film];"Print_List3")
GET PRINTABLE AREA(vLprint_height)
vLprinted_height:=0
ALL RECORDS([Film])

vSprint_area:="Header" `ヘッダエリアの印刷
$vLheight:=Print form([Film];"Print_List3";Form Header)
$vLheight:=21 `固定高
vLprinted_height:=vLprinted_height+$vLheight

While(Not(End selection([Film])))
   vSprint_area:="Detail" `詳細エリアの印刷
   $vLheight:=Print form([Film];"Print_List3";Form Detail)
 `詳細の計算はフォームメソッドで実行
   vLprinted_height:=vLprinted_height+$vLheight
   If(OK=0) `CANCEL がフォームメソッドで実行された
      PAGE BREAK
      vLprinted_height:=0
      vSprint_area:="Header" `ヘッダエリアの再印刷
      $vLheight:=Print form([Film];"Print_List3";Form Header)
      $vLheight:=21
      vLprinted_height:=vLprinted_height+$vLheight
      vSprint_area:="Detail"
      $vLheight:=Print form([Film];"Print_List3";Form Detail)
      vLprinted_height:=vLprinted_height+$vLheight
   End if
   NEXT RECORD([Film])
End while
PAGE BREAK `最後のページの印刷
```

Print_List3のフォームメソッド:

```
C_LONGINT($l;$t;$r;$b;$fixed_wdth;$exact_hght;$l1;$t1;$r1;$b1)
C_LONGINT($final_pos;$i)
C_LONGINT($detail_pos;$header_pos;$hght_to_print;$hght_remaining)

Case of
   :(vSprint_area="Detail") `詳細印刷実行中
      OBJECT GET COORDINATES([Film]Actors;$l;$t;$r;$b)
      $fixed_wdth:=$r-$l  ` Actors テキストフィールドサイズの計算
      $exact_hght:=$b-$t
      OBJECT GET BEST SIZE([Film]Actors;$wdth;$hght;$fixed_wdth)
 `内容に基づく最適なフィールドのサイズ
      $movement:=$hght-$exact_hght

      OBJECT GET COORDINATES([Film]Summary;$l1;$t1;$r1;$b1)
      $fixed_wdth1:=$r1-$l1  `Summaryテキストフィールドサイズの計算
      $exact_hght1:=$b1-$t1
      OBJECT GET BEST SIZE([Film]Summary;$wdth1;$hght1;$fixed_wdth1)
 `内容に基づく最適なフィールドのサイズ
      $movement1:=$hght1-$exact_hght1
      If($movement1>$movement)
 `最も高いフィールドの取得
         $movement:=$movement1
      End if

      If($movement>0)
         $position:=Get print marker(Form Detail)
         $final_pos:=$position+$movement
 `詳細マーカとそれより下のオブジェクトを移動
         SET PRINT MARKER(Form Detail;$final_pos;*)
 `テキストエリアのサイズ変更
         OBJECT MOVE([Film]Actors;$l;$t;$r;$hght+$t;*)
         OBJECT MOVE([Film]Summary;$l1;$t1;$r1;$hght1+$t1;*)

 `分離線のリサイズ
         OBJECT GET COORDINATES(*;"H1Line";$l;$t;$r;$b)
         OBJECT MOVE(*;"H1Line";$l;$final_pos-1;$r;$final_pos;*)
         For($i;1;4;1)
            OBJECT GET COORDINATES(*;"VLine"+String($i);$l;$t;$r;$b)
            OBJECT MOVE(*;"VLine"+String($i);$l;$t;$r;$final_pos;*)
         End for
      End if

 `利用可能なスペースの計算
      $detail_pos:=Get print marker(Form Detail)
      $header_pos:=Get print marker(Form Header)
      $hght_to_print:=$detail_pos-$header_pos
      $hght_remaining:=printing_height-vLprinted_height
      If($hght_remaining<$hght_to_print) `Insufficient height
         CANCEL `次のページに移動
      End if
End case
```

SET PRINT OPTION

SET PRINT OPTION (option ; value1 {; value2})

引数	型		説明
option	倍長整数, 文字	⇒	オプション番号
value1	倍長整数, 文字	⇒	オプションの値1
value2	倍長整数, 文字	⇒	オプションの値2

説明

SET PRINT OPTIONコマンドを使用し、プログラムから印刷オプションの値を変更することができます。プリントパラメーターを変更する他のコマンド (PRINT SETTINGS、> 引数を使用しないPRINT SELECTION) が呼び出されない限り、このコマンドを使用して定義された各オプションは、セッションの間、データベース全体に対して適用されます。

option引数を使用し、変更するオプションを指定することができます。"Print Options"テーマ内の定義済定数のいずれか、またはPDFオプションコード (WindowsのみでPDFCreatorドライバーで利用可能) を渡すことができます。

指定したoptionの新しい値は、value1と (オプションの) value2に渡します。渡す値の数と種類は、指定したオプションのタイプによって異なります。

option番号を使用する (定数)

以下の表でoptionとそれに対応するvalueを説明します:

option (定数)	value1	value2
1 (Paper option)	名前	-
	幅	高さ
2 (Orientation option)	1=縦方向, 2=横方向	-
3 (Scale option)	数値 (%)	-
4 (Number of copies option)	数値	-
5 (Paper source option)	Windowsのみ:	
	インデックス (数値)	-
8 (Color option)	Windowsのみ:	
	1=白黒, 2=カラー	-
9 (Destination option)	1=プリンタ,	-
	2=ファイル (PC)/PS (Mac),	アクセスパス
	3=PDFファイル,	アクセスパス
	5=スクリーン(Mac)	-
11 (Double sided option)	Windowsのみ:	
	0=片面 (標準)	-
	1=両面	とじしろ: 0=左 (デフォルト), 1=上
12 (Spooler document name option)	印刷するドキュメント名	-
13 (Mac spool file format option)	0=PDFモード, 1= PostScriptモード	-
14 (Hide printing progress option)	0=表示 (デフォルト), 1=非表示	-

- Paper option(1): PRINT OPTION VALUESコマンドを使用して取得できる、使用可能なすべての用紙タイプの名前一覧。
  value1に用紙の名前を渡すか (この場合value2は省略) 、またはvalue1に用紙の高さ、value2に用紙の幅を渡します。幅と高さは、画面のピクセル単位で表わさなくてはなりません。
- Orientation option(2): 1 (縦方向) または2 (横方向) のいずれかをvalue1に渡します。
- Scale option(3): 倍率 (%)をvalue1に渡します。プリンタには倍率を変更できないものがあるので注意が必要です。無効な値を渡すと、印刷時にこのプロパティが100%にリセットされます。
- Number of copies option(4): 印刷する部数をvalue1に渡します。
- Paper source option(5): 使用する用紙トレイを指定します。PRINT OPTION VALUESコマンドから返されるトレイ配列のインデックスに対応する数値を渡します。
  注: このオプションは、Windowsでのみ使用可能です。
- Color option(8): value1に、カラー処理用のモードを示すコードを渡します。1=黒白 、2=カラー。
  注: このオプションは、Windowsでのみ使用可能です。
- Destination option(9): value1に、印刷先のタイプを示すコードを渡します: 1=プリンタ、2=ファイル (PC) /PS (Mac) 、3=PDFファイル、5=スクリーン (Mac OS Xドライバオプション) 。
  value1が1または5以外の場合、出力結果となる文書へのアクセスパスをvalue2に渡します。別のパスが指定されるまで、セッションの間このパスが使用されます。出力先に同じ名前のファイルが既に存在する場合、新しいファイルと置き換えられます。
  注: Windowsでは、PDF Creatorプリンタードライバーがインストールされている場合のみ、印刷先として3 (PDFファイル) を指定できます。4D v12.2より、WindowsでPDF Creatorを使用して印刷を行う場合、4Dが制御を行うためにこの設定が必須となります。引数として (9;3;path) が渡されると、4Dは自動で、PDF Creatorに渡されたオプションコードを考慮した"サイレントな"PDF印刷を開始します (value2に空の文字列を渡した場合、または省略した場合、印刷時にファイルを保存ダイアログボックスが表示されます)。印刷後、カレントの設定が再ロードされます。これにより4DでのPDF印刷が簡略化され、マルチプラットフォームのコードが書けるようになります。
  (9;3;path)が渡されなかった場合、4Dは印刷を制御せず、PDF Creatorに渡されたオプションコードは無視されます。
- Double sided option(11): 0 (片面または標準) か1 (両面) のいずれかをvalue1に渡します。
  value1が1の場合、適用するとじしろをvalue2で指定できます: 0= 左とじ (デフォルト) または1= 上とじ
  注:このオプションは、Windowsでのみ使用可能です。
- Spooler document name option(12): value1に、スプーラ文書の一覧に表示したい印刷文書の名前を渡します。
  標準操作 (メソッドの場合はメソッド名、レコードの場合はテーブル名を使用等) を使用、または復帰するには、value1に空の文字列を渡します。
  警告: 新しい名前か空の文字列が渡されるまで、この命令文で指定した名前がセッションの印刷文書全体に対して使用されます。
  Mac spool file format option(13): value1に対し、印刷ジョブをPDFモード (デフォルト値) に設定する場合は0を渡し、印刷ジョブを"強制的に"PostScriptモードに設定する場合は1を渡します。このオプションはWindowsでは効果がありません。
  注: Mac OS X上では、デフォルトとして印刷はPDFとして行われます。しかしPDFプリントドライバでは、カプセル化されたPostScript情報を含むPICTピクチャがサポートされません。これらのピクチャは特に、ベクター描画ソフトにより生成されます。
  この問題を回避するため、このオプションを使用して、カレントセッション中にMac OSX上で用いる印刷モードを変更することができます。
  PostScriptモードで印刷を行うと、好ましくない副次的な結果をもたらす可能性があることを覚えておいてください。
- Hide printing progress option (14): value1に1を渡すと、進歩表示を隠すことができます。0を渡すと、再び表示されます (デフォルト)。このオプションはMac OS Xで特にPDF印刷の場合に役に立ちます。
  注: データベースダイアログボックス(インターフェースページ)には既に印刷進歩状況オプションがあります。しかしながら、それはアプリケーションにグローバルに適用されており、Mac OS Xですべてのウインドウを隠すというわけではありません。

このコマンドを使用して設定を行うと、4Dアプリケーション全体に対しセッションの間中、そのプリントオプションが保持されます。PRINT SELECTION、PRINT RECORD、Print form、QR REPORTコマンドおよびデザインモードを含めた4Dの印刷全般に対して、この設定が使用されます。

注:

- SET PRINT OPTIONコマンドを用いて設定したプリントオプションがリセットされないように、PRINT SELECTION、PRINT RECORD、PAGE BREAKコマンドでは、任意の引数 > を必ず使用してください。
- SET PRINT OPTIONコマンドは、ポストスクリプトプリンタでのみ動作します。

PDFオプションを使用する

option 引数でPDFオプションコードを使用できるようにするためには、4D環境にPDFCreatorドライバーがインストールされていなければなりません (詳細情報についてはWindowsにおけるPDFCreatorドライバーの統合を参照してください)。さらにオプションコードが効力を得るためには、以下の文を使用してPDF印刷の制御を有効にしなければなりません:

```
SET PRINT OPTION(Destination option;3;fileName)
```

そうしなければoptionコードは無視されます。

PDFoptionコードは2つの部分からなるテキストタイプのコードです。*OptionType*と*OptionName*を"*OptionType:OptionName*"のように組み合わせます。このコードの説明は以下の通りです:

- *OptionType* はネイティブなPDFCreatorオプションあるいは4D PDF管理オプションのいずれを設定するかを指定します。2つの値が受け入れられます:
  - **PDFOptions** = ネイティブオプション
  - **PDFInfo** = 内部オプション
- *OptionName* は設定するオプションを指定します (*OptionType* 値に基づきます)。
  - *OptionType* = **PDFOptions**の場合、*OptionName*には複数のPDFCreatorネイティブオプションのうち一つを渡せます。例えばUseAutosaveオプションは自動バックアップに影響します。このオプションを変更するには、option 引数に"PDFOptions:UseAutosave"を渡し、使用する値をvalue1引数に渡します。PDFCreatorネイティブオプションに関する完全な説明は、PDFCreatorドライバーの説明書を参照してください。
  - *OptionType* = **PDFInfo**の場合、*OptionName*には以下の特定のセレクターを渡せます:
    - **Reset print**: 特に無限ループから抜けるために、内部的な待ち状態をリセットするために使用します。この場合value1は使用しません。
    - **Reset standard options**: すべてのPDFCreatorオプションをデフォルト値にリセットするために使用します。印刷中の場合、デフォルト設定はその印刷が終了後に適用されます。この場合value1は使用しません。
    - **Start**: PDFCreatorスプーラーを開始または停止するために使用します。value1に0を渡すと停止、1を渡すと開始です。
    - **Reset options**: SET PRINT OPTIONコマンドおよび**PDFOptions**を使用して、セッションの開始以降変更されたすべてのオプションをリセットします。
    - **Version**: PDFCreatorドライバーの現在のバージョンを読み取るために使用します。このセレクターはGET PRINT OPTIONコマンドでのみ使用できます。番号はvalue1引数に返されます。
    - **Last error**: PDFCreatorドライバーから最後に返されたエラーを読みとるために使用します。このセレクターはGET PRINT OPTIONコマンドでのみ使用できます。エラー番号はvalue1引数に返されます。
    - **Print in progress**: PDFCreatorにより、4Dが印刷を待っているかどうかを知るために使用します。このセレクターはGET PRINT OPTIONコマンドでのみ使用できます。value1引数に1が返されると、4DはPDFCreatorを待っています。そうでなければ0が返されます。
    - **Job count**: 印刷キューにいくつのジョブがたまっているかを知るために使用します。このセレクターはGET PRINT OPTIONコマンドでのみ使用できます。ジョブ数はvalue1引数に返されます。
    - **Synchronous Mode**: 4Dが送信した印刷リクエストとPDFCreatorドライバー間の同期モードを設定するために使用します。4Dは印刷キュー内にある印刷ジョブの現在の状態に関する情報を取得できないので、このオプションを使用して、PDFCreatorドライバーが空き状態のときにのみジョブを送信することで、その実行をよりうまく制御できます。この場合、4Dはドライバーと同期されています。value1 に0を渡すと4Dは即座に印刷リクエストを送信します (デフォルト値)。そして1を渡すと4Dは同期を行い、他のリクエストを送信する前にドライバーがジョブを終了するのを待ちます。

注: それぞれの印刷後に、4DはPDFCreatorを使用する他のアプリケーションとの衝突を避けるために、自動でPDFCreatorドライバーの設定を以前のものに戻します。

例題

以下のメソッドはテーブル中の全レコードを印刷するために、PDFドライバーを有効にします。PDFはC:\Test\Test_PDF_X (Xはレコード位置番号) に書き出されます:

```
SET CURRENT PRINTER(PDFCreator Printer Name)
 // WindowsではPDFCreatorがインストールする仮想プリンターを選択
If(OK=1) // PDFCreatorが実際にインストールされていれば

   ALL RECORDS([Table_1])
   For($i;1;Records in selection([Table_1]))
      SET PRINT OPTION(Destination option;3;"C:\\Test\\Test_PDF_"+String($i))
 // Destination optionに3を指定することでPDFCreator印刷ジョブを起動
      PRINT RECORD([Table_1];*)
      NEXT RECORD([Table_1])
   End for
 // PDFCreatorドライバーのオプションをリセット
   SET PRINT OPTION("PDFInfo:Reset standard options";0)
End if
```

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

エラー管理

optionに渡した値が無効であるか、そのプリンタでoptionが利用できない場合、コマンドはエラーを返し (ON ERR CALLコマンドでインストールされたエラー管理メソッドを用いて、このエラーをとらえることができます) 、オプションの現在の値がそのまま保持されます。

SET PRINT PREVIEW

SET PRINT PREVIEW (preview)

引数	型	説明
preview	ブール	➡ スクリーンにプレビュー (TRUE), または プレビューしない (FALSE)

説明

SET PRINT PREVIEWは、プリントダイアログボックスのプレビュー設定のオン/オフをメソッドで切り替えるためのものです。previewにTrueを渡すとプレビューは有効になり、Falseを渡すと無効になります。この設定はプロセスに対してローカルであり、他のプロセスや他のユーザの印刷には影響を与えません。

例題

以下の例は、検索結果をスクリーン表示するために、まずプレビューを有効にし、それから無効に切り替えます。

```
QUERY([Customers])
If(OK=1)
   SET PRINT PREVIEW(True)
   PRINT SELECTION([Customers] ;*)
   SET PRINT PREVIEW(False)
End if
```

SET PRINTABLE MARGIN

SET PRINTABLE MARGIN (left ; top ; right ; bottom)

引数	型		説明
left	倍長整数	➡	左マージン
top	倍長整数	➡	上マージン
right	倍長整数	➡	右マージン
bottom	倍長整数	➡	下マージン

説明

SET PRINTABLE MARGIN コマンドを使用すると、Print formコマンドの使用時に、各種印刷マージンの値を設定することができます。

引数left, top, right, bottomには、以下の値のいずれかを渡すことができます:

- 0 = 用紙マージンを使用
- -1 = プリンタのマージンを使用
- 値 > 0 = ピクセル単位のマージン (72dpiの1ピクセルは約0.4 mm)

rightとbottom引数の値はそれぞれ、用紙の右および下の端に関連します。

Note: 印刷管理ならびに4D用語に関する詳細は、GET PRINTABLE MARGINコマンドの説明を参照してください。

デフォルトで、4Dはプリンタマージンに基づいて印刷を行います。SET PRINTABLE MARGINコマンドの実行後は、変更後のパラメータがセッション全体の同一プロセス内で維持されます。

例題 1

次の例題により、デッドマージンのサイズを取得することができます:

```
SET PRINTABLE MARGIN(-1;-1;-1;-1) `Sets the printer margin
GET PRINTABLE MARGIN($l;$t;$r;$b)
 `$l, $t, $r and $b correspond to the dead margins of the sheet
```

例題 2

次の例題により、用紙サイズを取得することができます:

```
SET PRINTABLE MARGIN(0;0;0;0) `Sets the paper margin
GET PRINTABLE AREA($height;$width)
 `For size A4: $height=842 ; $width=595 pixels
```

Subtotal

Subtotal (data {; pageBreak}) -> 戻り値

引数	型	説明
data	フィールド	→ 小計を求める数値型のフィールドまたは変数
pageBreak	倍長整数	→ 改ページを行うブレークレベル
戻り値	実数	↩ データの小計

説明

Subtotalは、現在または最後のブレークレベルにおけるdataの小計を返します。Subtotalは、ソートされたセレクションをPRINT SELECTIONコマンドで印刷する場合と、デザインモードでプリント...メニューから印刷を行う場合にのみ機能します。data引数のタイプは実数、整数、倍長整数のいずれかでなければなりません。フォームのブレークエリアに変数を配置し、Subtotalの結果を代入します。

警告: ブレーク処理を行ない、小計を計算するレポートを印刷する前に、BREAK LEVELとACCUMULATEコマンドを実行しなければなりません。このコマンドの最後の説明を参照してください。

Subtotalの2番目の引数 (オプション) で印刷中にページブレークを行わせることができます。pageBreakが0の場合、Subtotalは改ページを行いません。pageBreakが1の場合、Subtotalはレベル1の各ブレークに対して改ページを行います。pageBreakが2の場合、Subtotalはレベル1と2の各ブレークに対して改ページを行います。

Tip: 画面に表示された出力フォームからSubtotalを実行すると、エラーが発生し、フォームとエラーウィンドウの間で更新処理の無限ループを引き起こします。このループから抜けるには、エラーウィンドウのアボートボタンをクリックする際にAlt+Shiftキー (Windows) またはoption+shiftキー (Macintosh) を押します (何度か繰り返さなければならないかもしれません)。これによりフォームのウィンドウの更新が一時的に中断されます。エラーが再たび生成されるようにするために別のフォームを出力フォームとして選択してください。フォームを表示と印刷の両方で使用したい場合、デザインモードに移り、SubtotalをForm event=On Printing Breakという判定式の中に配置します。

例題

以下の例は、フォームのブレークエリア (B0、B0マーカの上のエリア) のオブジェクトメソッドです。vSalary変数はブレークエリアにあります。このブレークレベルが発生すると、変数に[Employees]Salaryフィールドの小計が代入されます。ブレーク処理は事前にACCUMULATEとBREAK LEVELコマンドを使用して有効にされていなければなりません。

```
Case of
   :(Form event=On Printing Break)
      vSalary:=Subtotal([Employees]Salary)
End case
```

フォームのヘッダとブレークエリアを使用したフォームのデザインに関するより詳しい情報は、4D Design Referenceマニュアルを参照してください。

フォームレポートにおけるブレーク処理の起動

ブレークを使用したレポートを生成するには、BREAK LEVELとACCUMULATEコマンドを呼び出して、レポート中のブレーク処理を有効にしなければなりません。

フォームレポートを印刷する前にこれらのコマンド両方を実行しなければなりません。フォームに値を表示するためにSubtotal関数は必要です。最低でも必要なブレークの数のレベルでソートしなければなりません。

BREAK LEVELとACCUMULATEを使用する際、レポートを印刷する処理は以下のようになります:
1. 印刷するレコードを選択する
2. ORDER BYでレコードをソートする。最低でもブレークレベル数でソートします。
3. BREAK LEVELとACCUMULATEを実行します。
4. PRINT SELECTIONでレポートを印刷します。

Subtotal関数はフォームに値を表示するために必要です。

命名セレクション

命名セレクション
CLEAR NAMED SELECTION
COPY NAMED SELECTION
CREATE SELECTION FROM ARRAY
CUT NAMED SELECTION
USE NAMED SELECTION

命名セレクション

命名セレクションは、複数のセレクションを同時に扱う簡単な方法を提供します。命名セレクションはプロセス中における、テーブルレコードの並び順付きリストです。この並び順付きリストに名前を付けてメモリに保持することができます。命名セレクションによって簡単にセレクションをメモリに置くことができます。命名セレクションはセレクションの並び順とカレントレコードをメモリに保持する簡単な手段を提供します。

以下のコマンドを使い、命名セレクション用いた作業を行うことができます:

- COPY NAMED SELECTION
- CUT NAMED SELECTION
- USE NAMED SELECTION
- CLEAR NAMED SELECTION
- CREATE SELECTION FROM ARRAY

命名セレクションは、COPY NAMED SELECTION、CUT NAMED SELECTIONまたはCREATE SELECTION FROM ARRAYコマンドで作成することができます。一般的に命名セレクションは、1つ以上のセレクションを用いて作業を行ったり、ソート済みのセレクションを保存して後で取り出したりするために使用します。プロセス内の各テーブルに対して複数の命名セレクションを持つことができます。命名セレクションをカレントセレクションとして再利用するには、USE NAMED SELECTIONを呼び出します。命名セレクションでの作業が終了したら、CLEAR NAMED SELECTIONを呼び出します。

Note: SET QUERY DESTINATION**(Into named selection;namedselection)** と (QUERYなどの) 検索コマンドを組み合わせても、命名セレクションを作成できます。詳細はSET QUERY DESTINATIONコマンドの説明を参照してください。

命名セレクションのスコープにはローカル、プロセス、インタープロセスがあります。

名前がドル記号 ($) で始まっている場合、命名セレクションはローカルです。名前が記号で始まっていない場合、それはプロセス命名セレクションです。名前が小なり大なり記号 (<>) で始まっている場合、それはインタープロセス命名セレクションです。

インタープロセス命名セレクションのスコープは、インタープロセス変数のそれと同じです。インタープロセス命名セレクションはすべてのプロセスからアクセス可能です。

4DのリモートモードとD Serverでは、インタープロセス命名セレクションはそれが作成されたマシン上のプロセスでのみ利用できます。インタープロセス命名セレクションを他のマシンから利用することはできません。

プロセス命名セレクションは、それが作成されたプロセス内およびサーバ上でのみ利用できます。

ローカル命名セレクションは、それが作成されたプロセス内でのみ利用可能で、サーバからは見えません。

警告: 命名セレクションを作成するには、テーブルのセレクションへのアクセスが必要となります。セレクションはサーバ上に保持され、ローカルプロセスはサーバデータへアクセスすることができないため、ローカルプロセス内で命名セレクションを使用してはいけません。

命名セレクションの可視性

以下の表は、スコープと作成された場所に基づき、命名セレクションがどの範囲で利用可能であるかの原則を説明しています:

	Client Process	Other client processes	Other clients	Server process	Other server processes
Creation in a client process					
$test	x				
test	x			x (Trigger)	
<>test	x	x			
Creation in a server process					
$test				x	
test				x	
<>test				x	x

命名セレクションとセット

セットと命名セレクションとの違いは、以下の通りです:

- 命名セレクションがレコードの並び順付きリストであるのに対して、セットは並び順を保持しません。
- セットは、テーブル内の各レコードに対して1ビットしか必要としないためメモリの面から見ると効率的です。命名セレクションは、セレクション内の各レコードに対して4バイトを必要とします。
- セットと違い、命名セレクションをディスクへ保存することはできません。
- セットには交差、結合、差異といった標準演算があるのに対して、命名セレクションは他の命名セレクションとの組み合わせ操作は行えません。

命名セレクションとセットには以下のような類似点があります:

- セットと同じように命名セレクションもメモリ上に存在します。
- 命名セレクションもセットもレコードの参照を格納します。レコードを変更または削除すると、命名セレクションやセットは無効になることがあります。
- セットと同じように命名セレクションは、その命名セレクションが作成された時点のカレントレコードを"記憶"します。

CLEAR NAMED SELECTION

CLEAR NAMED SELECTION (name)

引数	型		説明
name	文字	→	クリアする命名セレクション名

説明

CLEAR NAMED SELECTION は、命名セレクションnameをメモリから消去して、nameが使用していたメモリを解放します。CLEAR NAMED SELECTIONはテーブル、セレクション、およびレコードには影響しません。命名セレクションはメモリを使用するため、不要になったら消去する習慣をつけることをお勧めします。

CUT NAMED SELECTIONコマンドを使用して命名セレクションnameを作成し、USE NAMED SELECTIONコマンドで処理した場合には、命名セレクションnameは既にメモリ上にはありません。この場合は、CLEAR NAMED SELECTIONコマンドを使用する必要はありません。

⚙ COPY NAMED SELECTION

COPY NAMED SELECTION ({aTable ;} name)

引数	型	説明
aTable	テーブル	→ セレクションをコピーするテーブル, または 省略した場合デフォルトテーブル
name	文字	→ 作成する命名セレクションの名前

説明

COPY NAMED SELECTION はaTableのカレントセレクションを命名セレクションnameにコピーします。オプションtable引数が指定されていない場合は、そのプロセスのデフォルトテーブルを使用します。name引数にはセレクションのコピーが納められます。そのプロセスにおけるtableのカレントセレクションとカレントレコードは変更されません。

命名セレクションは実際にレコードを含むわけではなく、レコードへの並び順付き参照を含みます。各レコード参照はメモリを4バイト消費します。つまりCOPY NAMED SELECTIONコマンドを使用してセレクションをコピーすると、セレクション内のレコード数 x 4バイト分のメモリが必要となります。命名セレクションはメモリに置かれるため、命名セレクションに必要な分とプロセス内のテーブルのカレントセレクションに必要な分のメモリを確保しなければなりません。

nameが使用したメモリを解放するには、CLEAR NAMED SELECTIONコマンドを使用します

例題

以下の例では、[People]テーブルに未払いの送り状があるかどうかを調べています。セレクションをソートして保存します。請求書が未払いのレコードをすべて検索します。その後、そのセレクションを再利用してメモリ内の命名セレクションを消去します。ソートしたセレクションを後で使いたい場合には、命名セレクションを消去しなくても構いません:

```
ALL RECORDS([People])
  `ユーザにセレクションのソートを許可する
ORDER BY([People])
  ` 命名セレクションとしてソートしたセレクションを保存
COPY NAMED SELECTION([People];"UserSort")
  ` 未払いの請求書を検索
QUERY([People];[People]InvoiceDue=True)
  ` レコードがあれば
If(Records in selection([People])>0)
  ` ユーザに警告
   ALERT("Yes, there are overdue invoices on テーブル.")
End if
  ` ソート済みの命名セレクションを再利用
USE NAMED SELECTION("UserSort")
  ` メモリからセレクションを取り除く
CLEAR NAMED SELECTION("UserSort")
```

CREATE SELECTION FROM ARRAY

CREATE SELECTION FROM ARRAY (aTable ; recordArray {; selectionName})

引数	型		説明
aTable	テーブル	→	セレクションを作成するテーブル
recordArray	倍長整数, ブール配列	→	レコード番号の配列, または ブール配列 (True = レコードをセレクションに含める False = レコードをセレクションに含めない)
selectionName	文字	→	作成する命名セレクションの名前, または 引数を省略した場合、コマンドをカレント セレクションに適用する

説明

CREATE SELECTION FROM ARRAY コマンドは、以下の方法でselectionName命名セレクションを作成します:

- aTableのレコード番号値を納めた配列、または
- aTableのレコードごとに、レコードを含める (True)/含めない (False)の値を納めたブール配列。

selectionNameを省略した場合や空の文字列を渡した場合、コマンドはカレントセレクションに適用されます。結果カレントセレクションは更新されます。

このコマンドで倍長整数配列を使用すると、配列の各要素は作成されるselectionName内のレコードのレコード番号を表わします。レコード番号が正しくない (作成されていないレコード) 場合、エラー-10503が生成されます。

Note: 配列の中には、同じレコード番号が含まれないよう注意してください。そうでなければ、結果としてセレクションは不正確なものになります。

このコマンドでブール配列を使用する場合、配列のN番目の要素はレコード番号NがselectionNameに含まれる (True) か含まれないか (False) を示します。recordArrayの要素数はaTableのレコード数と等しくなければなりません。配列要素数がレコード数よりも少ない場合、配列によって定義されたレコードのみがセレクションとなります。

Note: ブール配列では、コマンドは配列要素0から (テーブルのレコード数) -1を使用します。

警告: 命名セレクションはメモリ内に作成、ロードされます。したがって、このコマンドを実行する前に十分なメモリがあることを確認してください。

エラー管理

レコード番号が不正 (レコードがまだ作成されていない) だった場合、エラー-10503が生成されます。このエラーはON ERR CALLでインストールされたエラー処理メソッドでとらえることができます。

CUT NAMED SELECTION

CUT NAMED SELECTION ({aTable ;} name)

引数	型	説明
aTable	テーブル	⇒セレクションをカットするテーブル, または 省略した場合デフォルトテーブル
name	文字	⇒作成する命名セレクションの名前

説明

CUT NAMED SELECTION は、命名セレクションnameを作成し、aTableのカレントセレクションをそこへ移します。このコマンドは、カレントセレクションをコピーするのではなく、移動する点がCOPY NAMED SELECTIONコマンドと異なります。

このコマンドを実行した後、カレントプロセスのaTableのカレントセレクションは空になります。そのため、CUT NAMED SELECTIONはレコードが修正されている最中は使用しないでください。

CUT NAMED SELECTIONはCOPY NAMED SELECTIONよりも効率的です。COPY NAMED SELECTIONでは選択したレコードの数x4バイトをメモリ内で複製します。CUT NAMED SELECTIONではリストの参照だけを移動します。

例題

以下のメソッドは、[Customers]テーブルのカレントセレクションを空にします:

```
CUT NAMED SELECTION([Customers];"ToBeCleared")
CLEAR NAMED SELECTION("ToBeCleared")
```

USE NAMED SELECTION

USE NAMED SELECTION (name)

引数	型	説明
name	文字	→ 使用する命名セレクション名

説明

USE NAMED SELECTION は、命名セレクションnameが属するテーブルのカレントセレクションを命名セレクションを使用して置き換えます。

命名セレクションを作成すると、その命名セレクションはカレントレコードを記憶します。USE NAMED SELECTIONコマンドはこのレコードの位置を取り出し、そのレコードを新しいカレントレコードにします。このコマンドはカレントレコードをロードします。nameを作成した後にカレントレコードが更新された場合、変更情報を失わないためにUSE NAMED SELECTIONコマンドを実行する前にそのレコードを保存してください。

- COPY NAMED SELECTIONコマンドを使用して命名セレクションnameを作成した場合、nameはそれが属するテーブルのカレントセレクションにコピーされます。nameは消去されるまでメモリに残ります。この命名セレクションを消去してnameはが使用しているメモリを解放するには、CLEAR NAMED SELECTIONコマンドを用います。
- CUT NAMED SELECTIONコマンドを使用してnameを作成した場合は、カレントセレクションがnameに設定され、nameはメモリから消去されます。

命名セレクションは、それが作成された時点でのレコードのセレクションを表わす点に注意してください。命名セレクションが表わしているレコードが変更されると、命名セレクションは正確ではなくなります。したがって、命名セレクションはあまり変更されないレコードの集まりを表わします。命名セレクションが無効になる原因はいくつかあります。例えば、命名セレクションのレコードの変更、削除、および命名セレクションを決定する条件の変更等があります。

変数

- CLEAR VARIABLE
- LOAD VARIABLES
- SAVE VARIABLES
- Undefined

CLEAR VARIABLE

CLEAR VARIABLE (variable)

引数	型		説明
variable	変数	⇒	クリアする変数

説明

CLEAR VARIABLE コマンドを使用して、variable をそのデフォルト値へ再設定します (文字列変数とテキスト変数は空の文字列に、数値変数には 0 に、配列はエレメントを空にします) 。しかし、変数はメモリに存在しています。

引数 variable に渡す変数は、プロセスまたはインタプロセス変数でなくてはなりません。

Note: プロセスの終了時、ユーザはプロセス変数を消去する必要はありません。4Dが自動的に消去します。

ドル記号 ($) で始まるローカル変数は CLEAR VARIABLE コマンドでは消去できません。その変数の属するメソッドの実行が完了した時点で、自動的に消去されます。

例題

フォーム上でユーザインターフェースの目的だけで、asMyDropDown ドロップダウンリストを使用するとします。つまり、データ入力中には配列を使用しますが、フォームから抜けた後はその配列が不要となります。したがって、On Unload イベントでこの配列を消去します。

```
  ` asMyDropDown ドロップダウンリストのオブジェクトメソッド
 Case of
    :(Form event=On Load)
  `配列を初期化
       ARRAY STRING(63;asMyDropDown;...)
  ` ...
    :(Form event=On Unload)
  `配列は不要
       CLEAR VARIABLE(asMyDropDown)
  ` ...
 End case
```

LOAD VARIABLES

LOAD VARIABLES (document ; variable {; variable2 ; ... ; variableN})

引数	型		説明
document	文字	→	4D変数を保存したドキュメント
variable	変数	→	値を受け取る変数

説明

LOAD VARIABLES コマンドは、documentによって指定されたドキュメントから1つまたはいくつかの変数を読み込みます。そのドキュメントはSAVE VARIABLES コマンドで作成されたものでなくてはなりません。

変数 variable 、variable2...variableN が作成されます。これらの変数が既に存在する場合、上書きされます。

documentに空の文字列を指定した場合、標準的なファイルを開くダイアログボックスが表示されます。ここで、ユーザは開くドキュメントを選択することができます。ドキュメントが選択されると、4Dシステム変数Documentにドキュメント名が入ります。

コンパイルされたデータベースでは、すべての変数はディスクから読み込まれた変数と同じタイプでなくてはなりません。

警告: このコマンドは、配列変数をサポートしません。新しく導入されたBLOBコマンドを使用してください。

例題

以下の例題を使用して、UserPrefsという名前のドキュメントから3つの変数を読み込みます。

```
 LOAD VARIABLES("User Prefs";vsName;vlCode;vgIconpicture)
```

システム変数およびセット

変数が正しくロードされるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

SAVE VARIABLES

SAVE VARIABLES (document ; variable {; variable2 ; ... ; variableN})

引数	型		説明
document	文字	→	変数を保存するドキュメントファイル
variable	変数	→	保存する変数

説明

SAVE VARIABLES コマンドを使用して、引数 document に渡した名前を持つドキュメントに 1つまたは複数の変数を保存します。

変数は同じタイプでなくてもかまいませんが、文字列、テキスト、実数、整数、倍長整数、日付、時間、ブール、またはピクチャのいずれかである必要があります。

引数 document に空の文字列を指定した場合、標準のファイルを保存ダイアログボックスが表示されます。ここで、ユーザは作成するドキュメントを選択することができます。この場合、ドキュメントが作成されると、4Dのシステム変数 Documentにドキュメント名が入ります。

変数が正常に保存されると、OK変数に1が代入されます。その他の場合は、0が代入されます。

Note: SAVE VARIABLES コマンドで変数をドキュメントファイルに保存する場合は、4D専用の内部フォーマットで保存します。したがって、保存した変数は LOAD VARIABLES コマンド以外では読み込むことができません。SAVE VARIABLES コマンドで作成したドキュメントは、RECEIVE VARIABLE または RECEIVE PACKET で読み込まないでください。

警告: このコマンドは、配列変数をサポートしません。BLOBコマンドを使用してください。

例題

以下の例題を使用して、UserPrefsという名前のドキュメントに3つの変数を保存します。

```
SAVE VARIABLES("User Prefs";vsName;vlCode;vgIconppicture)
```

システム変数およびセット

変数が正しく保存されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

Undefined

Undefined (variable) -> 戻り値

引数	型		説明
variable	変数	→	テストする変数
戻り値	ブール	↩	True = 変数は現在未定義である False = 変数は現在定義されている

説明

Undefined コマンドはvariable が定義されていない場合Trueを返します。variable が定義されている場合Falseを返します。変数が定義されるのは、コンパイラ命令で変数が作成された場合や値が変数に代入された場合です。その他の場合は定義されません。

データベースがコンパイルされると、UndefinedコマンドはすべてのFalseを返します。

例題

以下の例は、アプリケーションの特定モジュール用のメニュー項目が選択されたときに、プロセスの作成を管理します。プロセスが既に存在する場合、そのプロセスを前面に配置し、存在しない場合には、プロセスを開始します。このため、アプリケーションのモジュールごとにインタープロセス変数**<>PID_...**を保持します。このインタープロセス変数は、で初期化されます。

データベースの開発を実行する際、モジュールを新しく追加します。を修正する (対応する**<>PID_...** の初期化を追加するため) 代りに、データベースを再度開いてモジュールを追加するたびにすべてを再度初期化します。新規モジュールの追加を管理するには、Undefined コマンドを使用します。

```
  ` M_ADD_CUSTOMERS グローバルメソッド

 If(Undefined(<>PID_ADD_CUSTOMERS)) `このコードは開発の中盤で管理を実行する
    C_LONGINT(<>PID_ADD_CUSTOMERS)
    <>PID_ADD_CUSTOMERS:=0
 End if

 If(<>PID_ADD_CUSTOMERS=0)
    <>PID_ADD_CUSTOMERS:=New process("P_ADD_CUSTOMERS";64*1024;"P_ADD_CUSTOMERS")
 Else
    SHOW PROCESS(<>PID_ADD_CUSTOMERS)
    BRING TO FRONT(<>PID_ADD_CUSTOMERS)
 End if
  `Note: P_ADD_CUSTOMERS プロセスマスターメソッドは、<>PID_ADD_CUSTOMERS を終了時に0にセットする
```

文字列

- **文字参照記号**
- Change string
- Char
- Character code
- CONVERT FROM TEXT
- Convert to text
- Delete string
- Get localized string
- GET TEXT KEYWORDS
- Insert string
- Length
- Lowercase
- Match regex
- Num
- Position
- Replace string
- String
- Substring
- Uppercase
- ISO to Mac
- Mac to ISO
- Mac to Win
- Win to Mac

文字参照記号

概要

文字参照記号: [[...]]

文字参照記号は、文字列から一文字のみを参照するのに使用します。この構文は、テキスト変数、文字列変数、および文字フィールドの任意の場所の文字を指し示します。

注: 英語版の4DではMac OS上で、≤や≥記号が使用されていましたが、v13以降は[[と]]に統一されました。

文字参照記号が代入演算子 (:=) の左側にある場合、文字列内の指定した位置に文字を代入します。

以下の例は、vsNameが空の文字列ではない場合に、vsNameの最初の文字を大文字にします。

```
If(Length(vsName)#0)
   vsName[[1]]:=Uppercase(vsName[[1]])
End if
```

また、文字列参照記号が式に使用された場合、文字 (参照される) は1文字の文字列として返されます。

```
  ` 以下の例はvtTextの最後の文字が@であるかをテストします。
If(Length(vtText)#0)
   If(Character code(Substring(vtText;Length(vtText);1))=At sign)
  ` ...
   End if
End if

  ` 文字参照記号を使用し、よりシンプルに記述できます。
If(Length(vtText)#0)
   If(Character code(vtText[[Length(vtText)]])=At sign)
  ` ...
   End if
End if
```

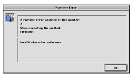
無効な文字列参照に関する注意

文字列参照記号を使用する際、配列の既存の要素を使用するのと同じ要領で、文字列内の既存の文字を使用しなければなりません。例えば、文字列変数の20文字目の文字を参照する場合、この変数は必ずすくなくとも20文字以上の長さがなくてはなりません。

- 長さが足りない場合、インタープリタモードでは構文エラーが発生します。
- 長さが足りない場合、コンパイルモード (オプション指定なし) では、例えば文字列やテキストの終わりを越えた位置に文字を書き込んだ場合にメモリ領域を破壊するおそれがあります。
- 長さが足りない場合、コンパイルモードで範囲チェックオプションをオンにしてある場合、例えば以下のようなコードを実行するとします。

```
  ` Very bad and nasty thing to do, boo!
vsAnyText:=""
vsAnyText[[1]]:="A"
```

すると、以下のようにランタイムエラーが表示されます。



```
A runtime error occured at line number:
3
When executing the method:
METHOD1

Invalid character reference.
```

例題

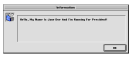
以下のプロジェクトメソッドは、文字列内の各単語の先頭の文字を大文字に変換した結果の文字列を返します。

```
  ` Capitalize text project method
  ` Capitalize text ( Text ) -> Text
  ` Capitalize text ( Source text ) -> Capitalized text

$0:=$1
$vlLen:=Length($0)
If($vlLen>0)
   $0[[1]]:=Uppercase($0[[1]])
   For($vlChar;1;$vlLen-1)
      If(Position($0[[$vlChar]];" !&()-{}:;<>?/,.=+*")>0)
         $0[[$vlChar+1]]:=Uppercase($0[[$vlChar+1]])
      End if
   End for
End if
```

使用例は以下の通り

```
ALERT(Capitalize text("hello, my name is jane doe and i'm running for president!"))
```

以下のように表示されます。



Hello, My Name Is Jane Doe And I'm Running For President!

Change string

Change string (source ; newChars ; where) -> 戻り値

引数	型		説明
source	文字	⇒	元の文字列
newChars	文字	⇒	新しい文字
where	倍長整数	⇒	入れ替え開始位置
戻り値	文字	↩	結果の文字列

説明

Change stringは、sourceの中の文字グループを修正したものを返します。whereで指定された位置から、newCharsでsourceを上書きします。

newCharsが空の文字列("")の場合、Change stringはsourceを変更しないで返します。Change stringは常にsourceと同じ長さの文字列を返します。whereがsourceの長さ以下の場合やsourceの長さ以上の場合、Change stringはsourceを返します。

Change stringは、文字を挿入しないで上書きするという点がInsert stringと異なります。

例題

Change stringの使用例を次に示します。結果を変数vtResultに代入します。

```
vtResult:=Change string("Acme";"CME";2) ` vtResultは"ACME"
vtResult:=Change string("November";"Dec";1) ` vtResultは"December"
```

Char

Char (charCode) -> 戻り値

引数	型		説明
charCode	倍長整数	→	文字コード
戻り値	文字	↩	文字コードによって表現された文字

説明

Char コマンド は文字コードがcharCodeである文字を返します。

charCode にはUTF-16値(1から65535の間) を渡します。

Tip: メソッド作成時、**Char** は通常キーボードから入力できない文字や、メソッドエディタでは編集コマンドとして解釈される文字を指定するために使用します。

例題

以下の例は、変数にキャリッジリターンを代入するためにChar関数を使用し、警告を表示します。

```
ALERT("Employees: "+String(Records in table([Employees]))+Char(Carriage return)+"Press OK to continue.")
```

Character code

Character code (character) -> 戻り値

引数	型		説明
character	文字	→	取得したい文字を得るためのコード
戻り値	倍長整数	↩	文字コード

説明

Character code コマンドは、character のUnicode UTF-16 コード(1から65535の間)を返します。

characterが1文字より多い場合、**Character code** は最初の文字だけをコードに変換します。

Character code の逆の変換を実行する関数が Char コマンドです。UTF-16コードが示す文字を返します。

例題 1

通常、大文字と小文字は同じものとして扱われますが、Character codeを使用すれば大文字と小文字を区別できます。以下の結果はTrueになります。

```
("A"="a")
```

一方、以下の結果はFalseになります。

```
(Character code("A")=Character code("a"))
```

例題 2

以下の例は、文字列"ABC"の最初の文字Aのコードを返します。

```
GetCode:=Character code("ABC") `GetCodeには 65が返されます。これは"A"の文字コードです
```

例題 3

以下の例は、キャリッジリターンとタブを検査します。

```
For($vlChar;1;Length(vtText))
   Case of
      :(vtText[[$vlChar]]=Char(Carriage return))
  ` 何らかの処理
      :(vtText[[$vlChar]]=Char(Tab))
  ` 何らかの処理
      :(...)
  ` ...
   End case
End for
```

サイズの大きなテキストに対して何度も実行する場合、以下のように記述した後コンパイルすると、この検査は高速に処理されます。

```
For($vlChar;1;Length(vtText))
   $vlCode:=Character code(vtText[[$vlChar]])
   Case of
      :($vlCode=Carriage return)
  ` 何らかの処理
      :($vlCode=Tab)
  ` 何らかの処理
      :(...)
  ` ...
   End case
End for
```

2 番目の例題が高速に処理される理由は2つあります。ループでは1文字だけが参照され、キャリッジリターンやタブを検査する際に、文字列の比較ではなく倍長整数による比較が行われています。CRやTAB等の一般的なコードを使用して作業する場合には、この手法を利用してください。

CONVERT FROM TEXT

CONVERT FROM TEXT (4Dtext ; charSet ; convertedBLOB)

引数	型		説明
4Dtext	文字	⇒	現在の4Dの文字セットで記述されているテキスト
charSet	文字, 倍長整数	⇒	文字セットの番号または名前
convertedBLOB	BLOB	⇐	変換されたテキストを含むBLOB

説明

CONVERT FROM TEXTコマンドは、現在の4Dの文字セットで記述されているテキストを、他の文字セットで記述されているテキストへ変換するために使用できます。

4Dtext 引数には変換するテキストを渡します。このテキストは、4Dが使用する文字セットで記述されています。4Dのバージョン11では、デフォルトでUnicode 文字セットが使用されています。

charSet には変換に使用する文字セットを渡します。セットの標準名 (例えば、"ISO-8859-1" や "UTF-8") 、またはMIBEnum識別子を渡すことができます。

CONVERT FROM TEXTとConvert to textコマンドでサポートされる文字セットのリストは以下のとおりです:

MIBEnum	名前
1017	UTF-32
1018	UTF-32BE
1019	UTF-32LE
1015	UTF-16
1013	UTF-16BE
1014	UTF-16LE
106	UTF-8
1012	UTF-7
3	US-ASCII
3	ANSI_X3.4-1968
3	ANSI_X3.4-1986
3	ASCII
3	cp367
3	csASCII
3	IBM367
3	iso-ir-6
3	ISO_646.irv:1991
3	ISO646-US
3	us
2011	IBM437
2011	cp437
2011	437
2011	csPC8CodePage437
2028	ebcdic-cp-us
2028	cp037
2028	csIBM037
2028	ebcdic-cp-ca
2028	ebcdic-cp-n
2028	ebcdic-cp-wt
2028	IBM037
2027	MacRoman
2027	x-mac-roman
2027	mac
2027	macintosh
2027	csMacintosh
2252	windows-1252
1250	MacCE
1250	x-mac-ce
2250	windows-1250
1251	x-mac-cyrillic
2251	windows-1251
1253	x-mac-greek
2253	windows-1253
1254	x-mac-turkish
2254	windows-1254
1256	x-mac-arabic
2256	windows-1256
1255	x-mac-hebrew
2255	windows-1255
1257	x-mac-ce
2257	windows-1257
17	Shift_JIS
17	csShiftJIS
17	MS_Kanji
17	Shift-JIS
39	ISO-2022-JP
39	csISO2022JP
2024	Windows-31J
2026	Big5
2026	csBig5
38	EUC-KR
38	csEUCKR
2084	KOI8-R
2084	csKOI8R
4	ISO-8859-1
4	CP819
4	csISOLatin1
4	IBM819
4	iso-ir-100
4	ISO_8859-1
4	ISO_8859-1:1987
4	l1
4	latin1
5	ISO-8859-2
5	csISOLatin2
5	iso-ir-101
5	ISO_8859-2
5	ISO_8859-2:1987
5	l2
5	latin2
6	ISO-8859-3
6	csISOLatin3
6	ISO-8859-3:1988
6	iso-ir-109
6	ISO_8859-3
6	l3
6	latin3
7	ISO-8859-4
7	csISOLatin4
7	ISO-8859-4:1988
7	iso-ir-110
7	ISO_8859-4
7	l4
7	latin4
8	ISO-8859-5
8	csISOLatinCyrillic
8	cyrillic
8	ISO-8859-5:1988
8	iso-ir-144
8	ISO_8859-5
9	ISO-8859-6
9	arabic
9	ASMO-708
9	csISOLatinArabic
9	ECMA-114
9	ISO-8859-6:1987
9	iso-ir-127
9	ISO_8859-6
10	ISO-8859-7
10	csISOLatinGreek
10	ECMA-118
10	ELOT_928
10	greek
10	greek8
10	iso-ir-126
10	ISO_8859-7
10	ISO_8859-7:1987
11	ISO-8859-8
11	csISOLatinHebrew
11	hebrew
11	iso-ir-138
11	ISO_8859-8
11	ISO_8859-8:1988
12	ISO-8859-9
12	csISOLatin5
12	iso-ir-148
12	ISO_8859-9
12	ISO_8859-9:1989
12	l5
12	latin5
13	ISO-8859-10
13	csISOLatin6
13	iso-ir-157
13	ISO_8859-10
13	ISO_8859-10:1992
13	l6
13	latin6
109	ISO-8859-13
111	ISO-8859-15
111	Latin-9
113	GBK
2025	GB2312
2025	csGB2312
2025	x-mac-chinesesimp
2025	x-mac-chinesesimp
57	GB_2312-80
57	csISO58GB231280

注: いくつかの行は同じMIBEnum識別子を持っています。これは文字セットが1つ以上の名前 (エイリアス) を持つためです。

文字セットの名前についての詳細は、以下のアドレスを参照してください:
http://www.iana.org/assignments/character-sets

コマンドを実行した後、convertedBLOB BLOBには変換されたテキストが返されます。このBLOBは、Convert to textコマンドで読み込むことができます。

注: IANAに定義されていない特別な文字セット (1258=x-mac-japanese) を使用することができます。これはWindows上ではコードページ10001に、Mac上ではkTextEncodingMacJapaneseにマップされています。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

Convert to text

Convert to text (BLOB ; charSet) -> 戻り値

引数	型		説明
BLOB	BLOB	→	特定の文字セットで記述されている テキストを含むBLOB
charSet	文字, 倍長整数	→	BLOB文字セットの番号または名前
戻り値	テキスト	↩	4Dの文字セットで表現されたBLOBの内容

説明

Convert to textコマンドは、blob引数に含まれているテキストを変換して、4Dの文字セットで記述されているテキストで返します。4Dはデフォルトで Unicode 文字コードを使用します。

charSetには変換に使用される、blobに含まれているテキストの文字セットを渡します。標準名やエイリアス名 (例えば"ISO-8859-1" や "UTF-8")、またはその識別子 (倍長整数) を指定できます。詳細はCONVERT FROM TEXTコマンドの説明を参照してください。

Convert to textはByte Order Marks (BOM) をサポートします。指定された文字セットがUnicodeタイプ (UTF-8, UTF-16またはUTF-32) であるとき、4Dは受信した最初のバイトでBOMの識別を試みます。BOMが検知できると、BOMは結果から取り除かれ、4DはcharSetではなくBOMが指定した文字セットを使用します。

システム変数およびセット

コマンドが正しく実行されるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

Delete string

Delete string (source ; where ; numChars) -> 戻り値

引数	型		説明
source	文字	→	文字を削除する文字列
where	倍長整数	→	削除開始位置
numChars	倍長整数	→	削除する文字数
戻り値	文字	↩	結果の文字列

説明

Delete stringは、whereからnumChars分の文字をsourceから削除した文字列を返します。

Delete stringは、以下のような場合にsourceと同じ文字列を返します。

- sourceが空の文字列の場合
- whereがsourceの長さより大きい場合
- numCharsがゼロ(0) の場合

whereが0より小さい場合、文字列の始めから文字が削除されます。

whereとnumCharsの和がsourceの長さと等しいかまたは大きい場合は、whereから文字列の最後まで文字を削除します。

例題

Delete stringの使用例を次に示します。結果を変数vtResultに代入します。

```
vtResult:=Delete string("Lamborghini";6;6) ` vtResultは"Lambo"
vtResult:=Delete string("Indentation";6;2) ` vtResultは"Indention"
vtResult:=Delete string(vtOtherVar;3;32000) ` vtResultはvtOtherVarの最初の2文字のみ
```

Get localized string

Get localized string (resName) -> 戻り値

引数	型	説明
resName	文字	→ resname属性値
戻り値	文字	↩ カレントランゲージで resNameによって指定された文字列の値

説明

Get localized stringコマンドは、resNameの属性によって指定された、カレントランゲージの文字列を返します。

このコマンドは、XLIFFのアーキテクチャ内だけで機能します。このタイプのアーキテクチャに関する詳細は、*Design Reference*マニュアルにあるXLIFFサポートを参照してください。

Note: Get database localizationコマンドを使用して、アプリケーションで使用するランゲージを調べることができます。

resNameに現在の対象ランゲージへの訳文を取得するための文字列リソース名を渡します。

Note: XLIFFは大文字小文字を区別します。

例題

以下は.xlfファイルの一部です:

```
<file source-language="en-US" target-language="ja">
[...]
   <trans-unit resname="Show on disk">
      <source>Show on disk</source>
      <target>ディスク上に表示</target>
   </trans-unit>
```

以下のステートメントを実行後:

```
 $JAvalue:=Get localized string("Show on disk")
```

...カレントランゲージが日本語の場合、$JAvalueには、"ディスク上に表示"が返されます。

システム変数およびセット

コマンドが正しく実行されるとOK変数に1が設定されます。resNameが見つからない場合、コマンドは空の文字列を返しOK変数に0が設定されます。

GET TEXT KEYWORDS

GET TEXT KEYWORDS (text ; arrKeywords {; *})

引数	型		説明
text	テキスト	→	元のテキスト
arrKeywords	テキスト配列	←	キーワードを受け取る配列
*	演算子	→	指定した場合、ユニークキーワード

説明

GET TEXT KEYWORDSコマンドはtextを個々の単語に分割し、arrKeywords配列の要素にして返します。

4Dはテキストを個々の単語に分割する際、キーワードインデックスを構築するのと同じアルゴリズムを使用します。これはICUライブラリに基づきます。どのようにテキストが単語に分割されるのかに関するより詳しい情報は以下のWebページを参照してください: http://userguide.icu-project.org/boundaryanalysis

注: ユーザーからのリクエストに基づき、フランス語とイタリア語に関しては例外が設けられました: 母音または"h"が続くアポストロフィ (') は単語区切り文字として扱われます。例えば文字列"L'homme"や"l'arbre"は"L'"+"homme"や"l'"+"arbre"などのように分割されます。

使用されるアルゴリズムはデータベース設定の**非文字・非数字のみをキーワード区切り文字とする**の設定により異なります (デザインリファレンスマニュアルのデータベース/データストレージページ参照)。

text 引数には単語に分割する元のテキストを渡します。スタイル付きテキストを渡すことができ、この場合スタイルタグは無視されます。

arrKeywordsにはテキストから取り出された単語のリストを受信するテキスト配列を渡します。

オプションの * 引数を渡すと、コマンドはarrKeywords配列内に重複しない値を返します。引数が省略された場合はデフォルトでテキストから取り出された単語がすべて格納され、複数回出現するものについてはその数だけ要素が作成されます。

このコマンドは、4Dと同じキーワードを使用しつつ、大量のテキストを含むレコードを検索する、簡素な方法を提供します。例えば"10,000 Jean-Pierre BC45"を含むテキストにおいて、このテキストがキーワード"10,000" + "Jean" + "Pierre" + "BC45"に分割されるとき、配列には4要素が含まれます。プログラムでこの配列をループし、%演算子を使用してこれらのキーワードを1つ以上含むレコードを検索することができます (例題参照)。

例題 1

検索エリアフォームにユーザーは1つ以上の単語を入力できます。ユーザーが検索実行を指示したら、プログラムはユーザーが入力した単語を最低1つ*MyField*フィールドに含むレコードを検索します。

```
  // vSearchはフォーム上の検索エリア変数名
 GET TEXT KEYWORDS(vSearch;arrSearch;*)
  // ユーザーが同じ単語を複数回入力した場合に備え*を渡す
 CREATE SET([MyTable];"Totalfound")
 $n:=Size of array(arrSearch)
 For($i;1;$n)
    QUERY([MyTable];[MyTable]MyField % arrSearch{$i})
    CREATE SET(([MyTable];"found")
    UNION("Totalfound";"found";"Totalfound")
 End for
 USE SET("Totalfound")
```

例題 2

先の例と同じフォームで、ユーザーが入力した単語を*MyField*フィールドにすべて含むレコードを検索します。

```
  // vSearchはフォーム上の検索エリア変数名
 GET TEXT KEYWORDS(vSearch;arrSearch;*)
 $n:=Size of array(arrSearch)
 QUERY([MyTable];[MyTable]ID >=0;*)
  // 検索初期化 = all records
 For($i;1;$n)
    QUERY([MyTable];&;[MyTable]MyField % arrSearch{$i};*)
  // 検索をスタック
 End for
 QUERY([MyTable]) //検索実行
```

例題 3

テキスト中の単語をカウント:

```
 GET TEXT KEYWORDS(vText;arrWords) // 全単語
 $n:=Size of array(arrWords)
 GET TEXT KEYWORDS(vText;arrWords;*) // 重複単語を除く
 $m:=Size of array(arrWords)
 ALERT("このテキストには"+String($n)+"個の単語が含まれ、重複を取り除くと"+String($m)+"個です。")
```

Insert string

Insert string (source ; what ; where) -> 戻り値

引数	型		説明
source	文字	→	文字列を挿入する文字列
what	文字	→	挿入する文字列
where	倍長整数	→	挿入する位置
戻り値	文字	↩	結果の文字列

説明

Insert stringは、sourceのwhereで指定された位置の前に、whatを 挿入した文字列を返します。

whatが空の文字列("") であれば、Insert stringはsourceを変更しないで返します。

whereが、sourceの長さよりも大きい場合は、whatをsourceの後ろに追加します。whereが1よりも小さい場合には、whatをsourceの前に挿入します。

Insert stringは、文字を上書きしないで挿入するという点がChange stringと異なります。

例題

Insert string関数の使用例を次に示します。結果を変数vtResultに代入します。

```
vtResult:=Insert string("The tree";" green";4) ` vtResultは"The green tree"
vtResult:=Insert string("Shut";"o";3) ` vtResultは"Shout"
vtResult:=Insert string("Indention";"ta";6) ` vtResultは"Indentation"
```

Length

Length (string) -> 戻り値

引数	型		説明
string	文字	→	長さを調べる文字列
戻り値	倍長整数	↩	文字列の長さ

説明

LengthはaStringの長さを知るために使用します。LengthはaString中の文字数を返します。

Note: 文字列に (無視可能な文字を含め) 何らかの文字が含まれているかどうかを判定したい場合、If (vtAnyText="")ではなくIf (Length(vtAnyText)=0)を使用しなければなりません。例えば文字列が無視可能な文字列であるChar(1)が1つで構成されている場合、Length(vtAnyText)は1を返しますが、vtAnyText=""はTrueを返します。

例題

Lengthの使用例を次に示します。結果を変数vlResultに代入します。コメントは、変数vlResultに代入される値についての説明です。

```
vlResult:=Length("Topaz") ` vlResultは5
vlResult:=Length("Citizen") ` vlResultは7
```

Lowercase

Lowercase (aString {; *}) -> 戻り値

引数	型	説明
aString	文字	→ 英小文字に変換する文字列
*	演算子	→ 渡した場合アクセントを保持
戻り値	文字	↩ 英小文字の文字列

説明

Lowercaseは、aStringを取りアルファベット文字列をすべて英小文字に変換して返します。

オプションの* 引数が渡されると、aStringに存在するアクセント符号付きの文字は、アクセント符号付きの英小文字で返されます。この引数が省略され、変換が実行されると、アクセント符号付きの文字は、デフォルトでそのアクセント記号を失います。

例題 1

以下の例は、Capsという名前の関数です。与えられた文字列の最初の文字を大文字にして返すものです。例えば、Caps("john") の結果は、"John" となります。

```
  ` Caps project method
  ` Caps ( 文字列 ) -> 文字列
  ` Caps ( Any text or 文字列 ) -> Capitalized text

 $0:=Lowercase($1)
 If(Length($0)>0)
    $0[[1]]:=Uppercase($0[[1]])
 End if
```

例題 2

この例では、引数が渡されたか、渡されないかに応じて取得された結果を比較します。

```
 $thestring:=Lowercase("DEJA VU") ` $thestringは"deja vu"
 $thestring:=Lowercase("DEJA VU";*) ` $thestringは"deja vu"
```

Match regex

Match regex (pattern ; aString ; start {; pos_found ; length_found}{; *}) -> 戻り値

引数	型		説明
pattern	テキスト	→	通常の式
aString	テキスト	→	検索が実行される文字列
start	倍長整数	→	aStringで検索が開始する位置
pos_found	倍長整数配列, 倍長整数変数	←	オカレンスの位置
length_found	倍長整数配列, 倍長整数変数	←	オカレンスの長さ
*	演算子	→	渡された場合、示された位置で検索するのみ
戻り値	ブール	↩	True = 検索がオカレンスを発見した場合 その他の場合はFalse

Match regex (pattern ; aString) -> 戻り値

引数	型		説明
pattern	テキスト	→	Regular expression (complete equality)
aString	テキスト	→	String in which search will be done
戻り値	ブール	↩	True = search has found an occurrence; Otherwise, False.

説明

Match regexコマンドを使用して、"正規表現" と呼ばれるメタ言語で合成された規則のセットと文字列が一致しているかを確認します。

正規表現を渡して、patternで検索します。これは特殊文字を用いて、文字列を説明するために使用される文字のセットから成ります。

aStringに正規表現で検索する文字列を渡します。

startでは、aString中で検索を開始する位置を渡します。

pos_foundとlength_foundが変数である場合、コマンド は位置とオカレンスの長さをこれらの変数に返します。配列を渡す場合、コマンドは位置とオカレンスの長さを配列の要素0に、正規表現によってキャプチャされたグループの位置と長さを続く要素に返します。

任意の* 引数 が渡されると、検索はstartで指定した位置から実行され、パターンにマッチしない場合でもそれ以降を検索しません。

検索がオカレンスを発見した場合、コマンドはTrueを返します。

正規表現に関する詳細は、以下の情報を参照してください:
http://ja.wikipedia.org/wiki/正規表現

pattern引数に渡す正規表現の構文についての詳細は、次のアドレスを参照してください:
http://www.icu-project.org/userguide/regexp.html

例題 1

完全に対等なものを検索する(単純なシンタックス):
vfound:=Match regex(pattern;mytext)

```
 QUERY BY FORMULA([Employees];Match regex(".*smith.*";[Employees]name))
```

例題 2

位置を用いてテキストで検索する
vfound:=Match regex(pattern;mytext; start; pos_found; length_found)
すべての$1タグを表示する例:

```
 vStart:=1
 Repeat
    vfound:=Match regex("<.*>";$1;vStart;pos_found;length_found)
    If(vfound)
       ALERT(Substring($1;pos_found;length_found))
       vStart:=pos_found+length_found
    End if
 Until(Not(vfound))
```

例題 3

括弧を使用したグループキャプチャのサポートを用いて検索する。正規表現では ( ) を使用してグループを定義する。
vfound:=Match regex(pattern;mytext; start; pos_found_array; length_found_array)

```
 ARRAY LONGINT(pos_found_array;0)
 ARRAY LONGINT(length_found_array;0)
 vfound:=Match regex("(.*)stuff(.*)";$1;1;pos_found_array;length_found_array)
 If(vfound)
    $group1:=Substring($1;pos_found_array{1};length_found_array{1})
    $group2:=Substring($1;pos_found_array{2};length_found_array{2})
 End if
```

例題 4

示されている位置のパターンの類似を制限しながら検索する
以前の2つの構文のうち１つの最後にスターを追加します。

```
 vfound:=Match regex("a.b";"---a-b---";1;$pos_found;$length_found)
  `True を返す
 vfound:=Match regex("a.b";"---a-b---";1;$pos_found;$length_found;*)
  `returns False を返す
 vfound:=Match regex("a.b";"---a-b---";4;$pos_found;$length_found;*)
  `returns True を返す
```

注: 返された位置と長さはUnicodeモードまたはテキストが7ビットASCIIタイプの場合のみその意味を持ちます。

エラー管理

エラーのイベントでは、コマンドはエラーを生成しますが、ON ERR CALLコマンドを用いてインストールされたメソッドで、これを検知することができます。

エラー管理

エラーのイベントでは、コマンドはエラーを生成しますが、ON ERR CALLコマンドを用いてインストールされたメソッドで、これを検知することができます。

Num

Num (expression {; separator}) -> 戻り値

引数	型		説明
expression	文字, ブール, 倍長整数	→	数値型を返したい文字列、または 0または1を返したいブール、または 数値式
separator	文字	→	小数区切り
戻り値	倍長整数	←	式引数の数値型

説明

Numコマンドは、expressionに渡した文字列、ブール、または数値式の数値型を返します。文字型の式を評価するために、オプションのseparator引数で小数区切り文字を指定できます。

文字列式

expressionが1つ以上のアルファベット文字からのみ構成される場合、Numは0を返します。expressionがアルファベット文字と数字を含む場合、Numはアルファベット文字を無視します。従って、Numは文字列"a1b2c3" を数値123に変換します。

Numが特別に扱う3つの指定された文字があります。(separator 引数が渡されていない場合) システムで定義された小数区切り文字、ハイフン"-" および "e"または"E" です。これらの文字は、数値表現のフォーマット文字として解釈されます。

- 小数区切りは小数点の位置として解釈され、数値文字列に埋め込まれていなければなりません。デフォルトでコマンドは、オペレーティングシステムに設定された小数区切りを使用します。separator引数を使用して、この文字を変更できます (以下参照) 。
- ハイフンは、数値や指数が負であることを意味します。ハイフンは負の数字文字列の前、または指数の場合"e"の後ろになければなりません。"e"をのぞきハイフンが数字の間にあると、それ以降の文字列は無視されます。例えば、Num("123-456")は123に、しかしNum("-9")は-9になります。
- eまたはEがあると、その右側の数字をすべて指数として解釈します。eは数字の文字列の間に置かなければなりません。Num("123e-2")は1.23になります。

文字列に"e"を含んでいる場合、MacとWindowsで異なる結果になる可能性があるので注意してください。

separator引数を使用して、expressionを評価するために使用するカスタム小数区切りを指定できます。評価される文字列が、システム演算子とは異なる小数区切りで表現されていると、コマンドは誤った結果を返します。この場合、separator引数を使用して正しい評価を取得できます。この引数が渡された場合、コマンドはこのシステム小数区切りを無視します。1つ以上の文字を渡すことができます。

Note: GET SYSTEM FORMATコマンドを使用して、カレント小数区切りおよびその他の特定のシステム引数を調べることができます。

ブール式
ブール式を渡した場合、Numは、式がTrueの場合1を返し、そうでなければ0を返します。

数値式
数値式をexpression引数に渡した場合、Numはexpression引数に渡された値をそのまま返します。これはポインタを使用するような汎用プログラミングで有効です。

例題 1

文字引数を渡した場合のNumの使用例を次に示します。結果を変数vResultに代入します。コメントは、変数vResultに代入される値についての説明です:

```
vResult:=Num("ABCD") ` vResultは0
vResult:=Num("A1B2C3") ` vResultは123
vResult:=Num("123") ` vResultは123
vResult:=Num("123.4") ` vResultは123.4
vResult:=Num("-123") ` vResultは -123
vResult:=Num("-123e2") ` vResultは -12300
```

例題 2

以下の例は、[Client]Debtを$1000と比較します。この比較に適用されるNumコマンドからは1または0が返されます。文字列に1や0を乗算するとその文字または空の文字が返されます。結果、[Client]Riskには"良い"または"悪い"が返されます。

```
 `顧客の負債額が、1000より小さいは「良い」
 `顧客の節額が、1000以上は「悪い」
[Client]Risk:=("良い"*Num([Client]Debt<1000))+("悪い"*Num([Client]Debt>=1000))
```

例題 3

この例は現在の小数区切りにより取得される結果を比較します:

```
$thestring:="33,333.33"
$thenum:=Num($thestring)
 `フランスのシステムでは、$thenumは、デフォルトで33,33333と等しい。
$thenum:=Num($thestring;".")
 `システムに関係なく、$thenum は正確に評価されます。
 `例えば、フランスのシステムでも 33 333,33となります。
```

Position

Position (find ; aString {; start {; lengthFound {; *}}}) -> 戻り値

引数	型		説明
find	文字	→	見つける文字列
aString	文字	→	調べる文字列
start	倍長整数	→	検索を開始する位置
lengthFound	倍長整数	←	調べた文字列の長さ
*	演算子	→	渡されると、文字コードに基づいて評価
戻り値	倍長整数	↩	最初に見つかった位置

説明

Positionコマンドは、aStringの中でfindが最初に現われる位置を返します。

aStringの中にfindが見つからない場合は、Positionはゼロ(0) を返します。

findが見つかると、aStringの中に検索文字列が最初に表示された文字位置を返します。

空のstringに対して空のfindを指定すると、Positionはゼロ(0) を返します。

デフォルトで、検索をaStringの最初の文字で開始します。任意のstart引数を使用して、aString中で検索を開始する文字位置を指定します。

lengthFound引数が渡されると、検索によって実際に見つかった文字列の長さを返します。この引数は、1つ以上の文字(例えばæとae、ßとss等) で書きこまれる文字を正確に表現するために必要となります。
* 引数が渡されたとき (以下参照) 、これらの文字は等しいと認識されない (æ # ae) ことに注意してください。このモードで、lengthFoundはいつもfindの長さと同じです (オカレンスが発見された場合)。

デフォルトでこのコマンドはグローバルな比較を行い、言語上の特性と、1つ以上の文字で記述される文字 (例 æ = ae) を考慮に入れます。他方、発音は区分せず (a=A, a=à 等) 、無視可能な文字は考慮されません (Unicodeの仕様)。無視可能な文字にはUnicodeの*C0 Control* サブセット (U+0000 ~ U+001F, Ascii制御文字セット) のすべての文字が含まれます (ただし印刷可能な文字 (U+0009 TAB, U+0010 LF, U+0011 VT, U+0012 FF そして U+0013 CR) は除きます。
この動作を変更するには、最後の引数にアスタリスク * を渡します。この場合、比較は文字コードベースで行われます。* 引数は以下のようなケースで必要となります:

- Char(1)など特別な文字を考慮に入れたい場合、
- 文字の評価で大文字小文字の区別やアクセント文字を考慮したい場合 (a#A, a#à 等)。

このモードでは、単語が書かれた方法のバリエーションが評価されないことに留意してください。

警告: Positionに対して@ワイルドカード記号を使用することはできません。例えば、findに"abc@" を渡すと、このコマンドは"abc" で始まる文字ではなく、単なる文字として"abc@" を検索します。

例題 1

Position関数の使用例を次に示します。結果を変数*vlResult*に代入します。コメントは、変数*vlResult*に代入される値についての説明です。

```
vlResult:=Position("ll";"Willow") ` vlResult gets 3
vlResult:=Position(vtText1;vtText2) ` Returns first occurrence of vtText1 in vtText2
vlResult:=Position("day";"Today is the first day";1) ` vlResult gets 3
vlResult:=Position("day";"Today is the first day";4) ` vlResult gets 20
vlResult:=Position("DAY";"Today is the first day";1;*) ` vlResult gets 0
vlResult:=Position("œ";"Bœuf";1;$length) ` vlResult =2, $length = 1
```

例題 2

次の例ではlengthFound引数を使用して、テキスト中に現れるすべての"aegis" を検索します。

```
$start:=1
Repeat
   vlResult:=Position("aegis";$text;$start;$lengthfound)
   $start:=$start+$lengthfound
Until(vlResult=0)
```

Replace string

Replace string (source ; oldString ; newString {; howMany}{; *}) -> 戻り値

引数	型		説明
source	文字	→	元の文字列
oldString	文字	→	置き換対象の文字列
newString	文字	→	置き換え後の文字列 (空文字の場合オカレンスは削除)
howMany	倍長整数	→	置き換え 省略時、すべてのオカレンスを置き換え
*	演算子	→	渡されると、文字コードに基づいて評価
戻り値	文字	↩	結果の文字列

説明

Replace stringは、sourceに存在するすべてのoldStringをnewStringでhowMany回数だけ置き換えます。

newStringが空の文字列 ("") の場合は、Replace stringはsourceの中のoldStringをすべて削除します。

howManyを指定した場合、Replace string関数はsourceの最初の文字から探して、その回数分だけoldStringを置き換えます。指定しない場合、発見したoldStringをすべて置き換えます。

oldStringが空の文字列の場合は、Replace stringはなにも変更せず、元の文字列を返します。

デフォルトでこのコマンドはグローバルな比較を行い、言語上の特性と、1つ以上の文字で記述される文字 (例 æ = ae) を考慮に入れます。他方、発音区分符号 (a=A, a=à等) は無視され、文字コードが9未満の制御コードは考慮されません (Unicodeの仕様)。
この動作を変更するには、最後の引数にアスタリスク * を渡します。この場合、比較は文字コードベースで行われます。* 引数は以下のようなケースで必要となります:

- Char(1)など特別な文字を考慮に入れたい場合、
- 文字の評価で大文字小文字の区別やアクセント文字を考慮したい場合 (a#A, a#a 等)。

このモードでは、単語が書かれた方法のバリエーションが評価されないことに留意してください。

例題 1

Replace stringの使用例を次に示します。結果を変数vtResultに代入します。コメントは、変数vtResultに代入される内容についての説明です。

```
vtResult:=Replace string("Willow";" ll";"d") `Resultは"Widow"
vtResult:=Replace string("Shout";"o";"") `Resultは"Shut"
vtResult:=Replace string(vtOtherVar;Char(Tab);",";*) `vtOtherVar の中の全てのタブをコンマ(.) に置き換える
```

例題 2

以下の例は、vtResultのテキストからキャリッジリターンとタブを取り除きます。

```
vtResult:=Replace string(Replace string(vtResult;Char(Carriage return);"";*);Char(Tab);"";*)
```

例題 3

この例では発音区分符号を区別するために、* 引数の使用する例を示します。

```
vtResult:=Replace string("Crème brûlée";"Brulee";"caramel") `Result gets "Crème caramel"
vtResult:=Replace string("Crème brûlée";"Brulee";"caramel";*) `Result gets "Crème brûlée"
```

String

String (expression {; format {; addTime}}) -> 戻り値

引数	型		説明
expression	式	→	文字列式を返したい式 (実数、整数、倍長整数、日付、時間、文字列、テキスト、ブールを指定可能)
format	文字, 倍長整数	→	表示フォーマット
addTime	時間	→	expressionが日付の時、文字列に追加する時間
戻り値	文字	↩	式の文字列式

説明

Stringコマンドは、expressionに渡した数値、日付、時間、文字列、またはブールを文字列に変換します。

オプションのformatを指定しない場合、適当なデフォルトの形式で文字列が返されます。formatを指定すると、結果の文字列は指定した形式になります。

オプションのaddTime引数は、日付に時間を複合フォーマットで追加します。この引数はepression引数が日付型の時にのみ使用できます (後述)。

数値式
expressionが数値（実数、整数、倍長整数）である場合、オプションで文字列フォーマットを渡すことができます。次に例を示します。

例題	結果	コメント
String(2^15)	"32768"	デフォルトフォーマット
String(2^15;"###,##0 Inhabitants")	"32,768 Inhabitants"	
String(1/3;"##0.00000")	"0.33333"	
String(1/3)	"0.33333333333333330000"	デフォルトフォーマット
String(Arctan(1)*4)	"3.141592653589790000"	デフォルトフォーマット
String(Arctan(1)*4;"##0.00")	"3.14"	
String(-1;"&x")	"0xFFFFFFFF"	
String(-1;"&$")	"$FFFFFFFF"	
String(0 ?+ 7;"&x")	"0x0080"	
String(0 ?+ 7;"&$")	"$80"	
String(0 ?+ 14;"&x")	"0x4000"	
String(0 ?+ 14;"&$")	"$4000"	
String(50,3;"&xml")	"50.3"	
String(Num(1=1);"True;;False")	"True"	
String(Num(1=2);"True;;False")	"False"	
String(Log(-1))	""	未定義の数値
String(1/0)	"INF"	正の無限数
String(-1/0)	"-INF"	負の無限数

フォーマットは、フォームの数値フォーマットと同じ方法で指定します。数値フォーマットの詳細については4D Design Referenceマニュアルの表示フォーマットを参照してください。カスタムフォーマットの書式名をformatに渡すことができます。カスタムフォーマットの名前は"|"で始めなければなりません。

日付式
expressionが日付の場合、デフォルトフォーマット（YY.MM.DD）を使用して文字列が返されます。format 引数には、以下で説明する定数のいずれか1つを渡すことができます (Date Display Formatsテーマ) 。

この場合addTime引数に時間を渡すことができます。この引数を使用すれば日付と時間を合成し、標準形式のタイムスタンプを生成することができます (ISO Date、ISO Date GMT、Date RFC 1123定数)。このフォーマットはXMLやWebの処理の際特に有効です。addTime引数はexpressionが日付型の場合のみ使用できます。

定数	型	値	コメント
Blank if null date	倍長整数	100	0の代わりに""
Date RFC 1123	倍長整数	10	Fri, 10 Sep 2010 13:07:20 GMT
Internal date abbreviated	倍長整数	6	Dec 29, 2006
Internal date long	倍長整数	5	December 29, 2006
Internal date short	倍長整数	7	2006/12/29
Internal date short special	倍長整数	4	06/12/29 (しかし 1986/12/29 または 2096/12/29)
ISO Date	倍長整数	8	2006-12-29T00:00:00
ISO Date GMT	倍長整数	9	2010-09-13T16:11:53Z
System date abbreviated	倍長整数	2	
System date long	倍長整数	3	
System date short	倍長整数	1	

以下は今日が2006/12/29の場合の例です。

```
$vsResult:=String(Current date) // $vsResultは"06/12/29"
$vsResult:=String(Current date;Internal date long) // $vsResultは"December 29, 2006"
$vsResult:=String(Current date;ISO Date GMT) // $vsResultは日本の場合"2006-12-28T15:00:00Z"
```

日付/時間複合フォーマットに関するメモ:

- ISO Date GMTフォーマットはISO8601標準に対応します。日付と時間を含みタイムゾーン (GMT) を考慮します。
  ```
  $mydate:=String(Current date;ISO Date GMT;Current time) // 2010-09-13T16:11:53Zを返します。
  ```

  上記の例の最後の"Z"はGMTフォーマットの終了を表します。
  addTime引数を渡さない場合、ローカルタイムの午前0時として扱いGMTに変換します (例題参照)。そのため日付がローカルタイムに対して前後することがあります。
  ```
  $mydate:=String(Current date;ISO Date GMT) // 2010-09-12T15:00:00Zを返す
  ```
- ISO DateフォーマットはISO Date GMTフォーマットと同様に日付と時間を含みますが、タイムゾーンを考慮しません。当初よりこのフォーマットはISO8601標準に準拠しておらず、非常に特殊な目的のために予約されたものです。
  ```
  $mydate:=String(!13/09/2010!;ISO Date) // 2010-09-13T00:00:00 を返す(タイムゾーンに依存しない)
  &NBSP;&NBSP;&NBSP;&NBSP;$mydate:=String(Current date;ISO Date;Current time) // 2010-09-13T18:11:53 を返す
  ```
- Date RFC 1123フォーマットは日付と時間の組み合わせをRFC 822と1123で定義された標準に基づきフォーマットします。このフォーマットはたとえばHTTPヘッダーでcookieの有効期限を設定する際に必要となります。
  ```
  $mydate:=String(Current date;Date RFC 1123;Current time) // Fri, 10 Sep 2010 13:07:20 GMTを返す
  ```

  表現される日時は、タイムゾーンが考慮されるためローカルのタイムゾーンにより日付が前後にずれることになります。日付のみを渡すと、コマンドはローカルタイムの00:00をGMT時間で表現して返します:
  ```
  $mydate:=String(Current date;Date RFC 1123) // Thu, 09 Sep 2010 15:00:00 GMTを返す
  ```

時間式
expressionが時間の場合、デフォルトフォーマット(HH:MM:SS)を使用して文字列が返されます。format 引数には、以下の表に示す定数のいずれか1つを渡すことができます (Time Display Formatsテーマ) 。

定数	型	値	コメント
Blank if null time	倍長整数	100	0の代わりに""
HH MM	倍長整数	2	01:02
HH MM AM PM	倍長整数	5	1:02 AM
HH MM SS	倍長整数	1	01:02:03
Hour Min	倍長整数	4	1時2分
Hour Min Sec	倍長整数	3	1時2分3秒
ISO Time	倍長整数	8	0000-00-00T01:02:03
Min Sec	倍長整数	7	62分3秒
MM SS	倍長整数	6	62:03
System time long	倍長整数	11	1:02:03 AM HNEC (Macのみ)
System time long abbreviated	倍長整数	10	1:02:03 AM (Macのみ)
System time short	倍長整数	9	01:02:03

Notes:

- ISO DateフォーマットはISO8601標準に対応し、日付と時間を含みます。このフォーマットは日付と時間の複合をサポートしないため、日付部分は0で埋められます。このフォーマットはローカル時間を表します。
- Blank if null**定数**はフォーマットに他のフォーマットに加算して使用しなければなりません。追加することにより、Null値の場合、4Dは0の代わりに空の文字列を返します。

以下の例は、現在時刻が5:30 PM45秒であるものとします。

```
$vsResult:=String(Current time) // $vsResultは"17:30:45"
$vsResult:=String(Current time;Hour Min Sec) // $vsResultは"17時30分45秒"
```

文字列式
expressionが文字列またはテキスト型の場合、コマンドは引数に渡した値と同じ値を返します。これは特にポインタを使用している汎用プログラミングで有効です。
この場合、format引数は渡されても無視されます。

ブール式
expressionがブール 型の場合、コマンドはアプリケーションのランゲージに文字列 "True" または"False"を返します(例えば、4Dのフランス語バージョンでは、"Vrai"または"Faux")。
この場合、format引数は渡されても無視されます。

Substring

Substring (source ; firstChar {; numChars}) -> 戻り値

引数	型		説明
source	文字	➡	一部を取り出す文字列
firstChar	倍長整数	➡	最初の文字位置
numChars	倍長整数	➡	取り出す文字列の長さ
戻り値	文字	↩	文字列の一部

説明

Substringコマンドは、firstCharとnumCharsで指定した部分文字列をsourceから取り出して返します。

firstCharには文字列の中で取り出す文字列の最初の文字の位置を指定し、numCharsには取り出す文字列の長さを指定します。

firstCharとnumCharsの和が、文字列自身の文字数よりも大きい場合やnumCharsを指定しない場合には、先頭文字位置以降の文字列をすべて取り出します。firstCharが文字列の長さより大きいと、Substringは空の文字列("") を返します。

例題 1

Substringコマンドの使用例を次に示します。結果を変数 vsResultに代入します。コメントは、変数 vsResultに代入される内容についての説明です。

```
vsResult:=Substring("08/04/62";4;2) ` vsResultは"04"
vsResult:=Substring("Emergency";1;6) ` vsResultは"Emerge"
vsResult:=Substring(var;2) `vsResult は一文字目を除いた全て
```

例題 2

以下のプロジェクトメソッドは、テキスト(最初の引数で指定) 中に見つかった段落を文字列またはテキスト配列(2番目の引数としてポインタを渡す) に追加します。

```
  ` EXTRACT PARAGRAPHS
  ` EXTRACT PARAGRAPHS ( text ; Pointer )
  ` EXTRACT PARAGRAPHS ( Text to parse ; -> 文字列配列 )

C_TEXT($1)
C_POINTER($2)

$vlElem:=Size of array($2->)
Repeat
   $vlElem:=$vlElem+1
   INSERT IN ARRAY($2->;$vlElem)
   $vlPos:=Position(Char(Carriage return);$1)
   If($vlPos>0)
      $2->{$vlElem}:=Substring($1;1;$vlPos-1)
      $1:=Substring($1;$vlPos+1)
   Else
      $2->{$vlElem}:=$1
   End if
Until($1="")
```

Uppercase

Uppercase (aString {; *}) -> 戻り値

引数	型		説明
aString	文字	→	英大文字にする文字列
*	演算子	→	渡されると、アクセント符号を保持
戻り値	文字	↩	英大文字に変換した文字列

説明

Uppercaseは、aStringを取りアルファベット文字列をすべて英大文字に変換して返します。

オプションの* 引数が渡されると、aStringに存在するアクセント符号付きの文字を、アクセント符号付きの英小文字で返します。この引数が省略され変換が実行されると、アクセント符号付きの文字は、デフォルトでそのアクセント符号を失います。

例題 1

この例では、引数が渡されたか、渡されないかに応じて取得された結果を比較します。

```
 $thestring:=Uppercase("helene") ` $thestringは"HELENE"
 $thestring:=Uppercase("helene";*) ` $thestringは"HELENE"
```

例題 2

Lowercaseの例を参照してください。

ISO to Mac

ISO to Mac (text) -> 戻り値

引数	型		説明
text	文字	→	標準的なWeb文字セットで表されたテキスト
戻り値	文字	↩	Mac OS ASCIIマップで表されたテキスト

互換性に関するメモ

データベースがASCII互換モードで実行された場合のみ、このコマンドは機能します。Unicodeモードでは、コマンドは何もしません(textの文字列は修正なしで返されます)。その為、4Dのバージョン11以降、このコマンドはあまり用いられていませんので、その使用はもはやお勧めできません。CONVERT FROM TEXTコマンド、またはConvert to textコマンドを用いて文字列を変換することをお勧めします。

説明

ISO to Macコマンドは、ISO Latin-1文字マップを使用して表されたtextと同等のテキストをMacintosh ASCIIマップを使用して表して返します。

通常、このコマンドを使用する必要はありません。

このコマンドには、ISO Latin-1文字マップを使用して表したテキスト引数が必要です。

4Dは、Webブラウザとの間で送受信する文字を変換します。その結果、操作するテキスト値は、Web接続プロセスの内部では、常にMacintosh ASCIIマップを使用して表されます。

通常、Windows上で実行している場合には、ASCIIコードを変換する必要はありません。ASCII互換モードで(非Unicode)、4D とWindowsの間でテキストをコピーする、または貼り付けたりする場合や、データを読み込む、または書き出す場合には、4Dは自動的にこのような変換 を実行します。ただし、SEND PACKETコマンドやRECEIVE PACKETコマンド等の、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、4DはASCII変換を実行しません。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上およびWindows上の双方ともにMac OSエンコードです。詳細については、付録のの節を参照してください。

Windowsでは、この場合には、出力フィルタASCIIマップを使用して文字をフィルタしないでおく必要があります。

したがって、プラットフォームが何であっても、RECEIVE PACKETコマンドを使用してディスクからISO Latin-1によるHTMLドキュメントを読み取りしたい場合には、ISO to Macを使用して対象となるテキストを変換するだけですみます。これが、ISO to Macコマンドの主要な目的です。

例題

以下のコードの行では、vtSomeTextに格納されている(仮想の)ISO Latin-1でコード化されたテキストを、Macintoshでコード化されたテキストに変換しています。

```
RECEIVE PACKET($vhDocRef;vtSomeText;16*1024) `ISO Latin-1 HTMLドキュメントをテキストから読み取る
vtSomeText:=ISO to Mac(vtSomeText)
```

Mac to ISO

Mac to ISO (text) -> 戻り値

引数	型		説明
text	文字	→	Mac OS ASCIIマップで表されたテキスト
戻り値	文字	←	標準的なWeb文字セットで表されたテキスト

互換性に関するメモ

データベースがASCII互換モードで実行された場合のみ、このコマンドは機能します。Unicodeモードでは、コマンドは何もしません(textの文字列は修正なしで返されます) 。その為、4Dのバージョン11以降、このコマンドはあまり用いられていませんので、その使用はもはやお勧めできません。CONVERT FROM TEXTやConvert to textコマンドを用いて文字列を変換することをお勧めします。

説明

Mac to ISOコマンドは、Web文字テーブルを使用して表されたtextを、4Dアプリケーションの環境設定のWeb/オプションページの**スタンダードセット**メニューで指定された文字セットで返します。

通常は、このコマンドを使用する必要はありません。

このコマンドには、Mac OS ASCIIマップを使用して表したテキスト引数が必要です。

4Dは、Webブラウザとの間で送受信する文字を変換します。その結果、操作するテキスト値は、Web接続プロセスの内部では、常にMac OS ASCIIマップを使用して表されます。

通常、Windows上で実行している場合には、このコマンドを使用してASCIIコードを変換する必要はありません。ASCII互換モード(非 Unicode) で、4DとWindowsの間でテキストをコピーする、または貼り付けたりする場合や、データを読み込む、または書き出す場合には、4Dは自動的にこのような変換を実行します。ただしSEND PACKETやRECEIVE PACKETコマンド等の、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、ASCII変換を明示的に実行する必要があります。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上およびWindows上の双方ともにMac OSエンコードです。詳細については、付録のASCIIコードを参照してください。

Windowsでは、この場合には、出力フィルタASCIIマップを使用して文字をフィルタしないでおく必要があります。

したがって、プラットフォームが何であっても、ISO Latin-1を使用してディスク上にHTMLドキュメントを作成したい場合には、**Mac to ISO**を使用して対象となるテキストを変換するだけですみます。これが**Mac to ISO**コマンドの主要な目的です。

例題 1

以下のコードの行では、vtSomeTextに格納されている(仮想の)Mac OSでコード化されたテキストを、ISO Latin-1でデコードされたテキストに変換しています。

```
vtSomeText:=Mac to ISO(vtSomeText)
```

例題 2

4D Web Serverアプリケーションの開発中にHTMLドキュメントを作成し、後でWEB SEND FILEコマンドを使用してイントラネットまたはインターネット経由で送信しようとしています。これらのHTMLドキュメントの一部は他のドキュメントに参照やリンクを持っています。

以下のプロジェクトメソッドでは、引数として受信したWindowsからのHTMLベースのパス名やMacintoshのパス名を求めています。

```
  ` HTML Pathnameプロジェクトメソッド
  ` HTML Pathname ( Text ) -> Text
  ` HTML Pathname ( Native File Manager Pathname ) -> HTML HTMLパス名

C_TEXT($0;$1)
C_LONGINT($vlChar;$vlAscii)
C_STRING(31;$vsChar)

$0:=""
If(On Windows)
   $1:=Replace string($1;"\";"/")
Else
   $1:=Replace string($1;":";"/")
End if
$1:=Mac to ISO($1)
For($vlChar;1;Length($1))
   $vlAscii:=Character code($1[[$vlChar]])
   Case of
      :($vlAscii>=127)
         $vsChar:="%"+Substring(String($vlAscii;"&$");2)
      :(Position(Char($vlAscii);":<>&%= "+Char(34))>0)
         $vsChar:="%"+Substring(String($vlAscii;"&$");2)
      Else
         $vsChar:=Char($vlAscii)
   End case
   $0:=$0+$vsChar
End for
```

注: On Windowsプロジェクトメソッドはシステムドキュメントの章の中で示されています。

On Windowプロジェクトメソッドをデータベースに作成した後、特定のディレクトリにあるドキュメントへのFTPリンクのリストを追加したい場合には、以下のように記述できます。

```
  `On Startupデータベースメソッド内でインタープロセス変数を設定する
<>vsFTPURL:="ftp://123.4.56.78/Spiders/"
<>vsFTPDirectory:="APS500:Spiders:" `これはMac OSファイル管理のパス名
  ` ...

  ` ...
ARRAY TEXT($asDocuments;0)
DOCUMENT LIST(...;$asDocuments)
$vlNbDocuments:=Size of array($asDocuments)
jsHandler:=...
For($vlDocument;1;$vlNbDocuments)
   vtHTMLCode:=vtHTMLCode+"<p><a href=\""<>vsFTPURL+HTML Pathname(Substring($1+$asDocuments{$vlDocument};
   Length(<>vsFTPDirectory)+1))+"\""+jsHandler+"> "+$asDocuments{$vlDocument}+"</a></p>\r"
End for
```

Mac to Win

Mac to Win (text) -> 戻り値

引数	型		説明
text	文字	→	Mac OS ASCIIマップで表されたテキスト
戻り値	文字	↩	Windows ANSIマップで表されたテキスト

互換性に関するメモ

データベースがASCII互換モードで実行された場合のみ、このコマンドは機能します。Unicodeモードでは、コマンドは何もしません(textの文字列は修正なしで返されます) 。その為、4Dのバージョン11以降、このコマンドはあまり用いられていませんので、その使用はもはやお勧めはできません。CONVERT FROM TEXTコマンド、またはConvert to textコマンドを用いて文字列を変換することをお勧めします。

説明

Mac to Win コマンドは、Macintosh ASCIIマップを使用して表された、textに渡さたテキストと同等のWindows ANSIマップを使用したテキストに変換して返します。

この関数には、Macintosh ASCIIマップを使用して表したテキスト引数が必要です。

通常、Windows上で実行している場合には、このコマンドを使用してASCIIコードを変換する必要はありません。ASCII互換モード(非 Unicode) 、4DとWindowsの間でテキストをコピーする、または貼り付けたりする場合や、データを読み込む、または書き出す場合には、4Dは自動的にこのような変換を実行します。ただし、SEND PACKETコマンドやRECEIVE PACKETコマンド等の、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、ASCII変換を明示的に実行する必要があります。これが、Mac to Win関数の主要な目的です。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上およびWindows上の双方ともにMac OSエンコードです。詳細については、付録のを参照してください。

Note: このコマンドはCR（キャリッジリターン）文字をCRLF（キャリッジリターン+改行、文字コード13+10）に置き換えます。したがって文字数が元より多くなります。

例題

Windows上でSEND PACKETコマンドを使用してドキュメントに文字を書き込む時に、出力ASCIIマップを使用してMacintoshからWindowsへの文字のフィルタリングを実行(USE CHARACTER SETを参照) していない場合には、手動でテキストをMacintoshからWindowsに変換する必要があります。この操作は、以下のように実行します。

```
  ` ...
 SEND PACKET($vhDocRef;Mac to Win(vtSomeText))
  ` ...
```

Win to Mac

Win to Mac (text) -> 戻り値

引数	型		説明
text	文字	→	Windows ANSIマップで表されたテキスト
戻り値	文字	↩	Macintosh ASCIIマップで表されたテキスト

互換性に関するメモ

データベースがASCII互換モードで実行された場合のみ、このコマンドは機能します。Unicodeモードでは、コマンドは何もしません(textの文字列は修正なしで返されます) 。その為、4Dのバージョン11以降、このコマンドはあまり用いられていませんので、その使用はもはやお勧めはできません。CONVERT FROM TEXTコマンド、またはConvert to textコマンドを用いて文字列を変換することをお勧めします。

説明

Win to Macコマンドは、Windows ANSIマップを使用して表されるtextと同等のテキストをMacintosh ASCIIマップを使用して返します。

このコマンドには、Windows ANSIマップを使用して表したテキスト引数が必要です。

通常、Windows上で実行している場合には、このコマンドを使用してASCIIコードを変換する必要はありません。ASCII互換モード(非Unicode) で4DとWindowsの間でテキストをコピーする、または貼り付けたりする場合や、データを読み込む、または書き出す場合には、4Dは自動的にこのような変換を実行します。ただし、SEND PACKETコマンドやRECEIVE PACKETコマンド等の、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、ASCII変換を明示的に実行する必要があります。これが、Win to Mac コマンドの主要な目的です。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上およびWindows上の双方ともにMac OSエンコードです。詳細については、付録のを参照してください。

Note: このコマンドはCRLF（キャリッジリターン+改行、文字コード13+10）をCR（キャリッジリターン）文字に置き換えます。したがって文字数が元より少なくなります。

例題

WindowsでRECEIVE PACKETコマンドを使用してドキュメントから文字を読み込む時に、入力ASCIIマップを使用してWindowsからMacintoshへの文字のフィルタリングを実行（USE CHARACTER SETコマンドを参照）していない場合には、手動でテキストをWindowsからMacintoshに変換する必要があります。この操作は、以下のように実行します。

```
  ` ...
RECEIVE PACKET($vhDocRef;vtSomeText;16*1024)
vtSomeText:=Win to Mac(vtSomeText)
  ` ...
```

日付と時間

- Add to date
- Current date
- Current time
- Date
- Day number
- Day of
- Milliseconds
- Month of
- SET DEFAULT CENTURY
- Tickcount
- Time Updated 14.0
- Time string
- Year of

Add to date

Add to date (date ; years ; months ; days) -> 戻り値

引数	型		説明
date	日付	→	年月日を加算する日付
years	倍長整数	→	日付に加算する年
months	倍長整数	→	日付に加算する月
days	倍長整数	→	日付に加算する日
戻り値	日付	↩	結果の日付

説明

Add to date コマンドは、dateにyears、months、days を加算し、その結果の日付を返します。

通常、任意の日付に日を追加する場合、を使用しますが、Add to dateは（+ 日付 演算子を使用する場合のように）1ケ月の日数やうるう年の取り扱い方法を気にすることなく月や年を素早く加算することができます。

例題

```
  ` この行は、現在の月日の一年後の日付を計算しています
 $vdInOneYear:=Add to date(Current date;1;0;0)

  ` この行は、現在の日付の来月を計算しています
 $vdNextMonth:=Add to date(Current date;0;1;0)

  ` この行は$vdTomorrow:=Current 日付+1と同等です
 $vdTomorrow:=Add to date(Current date;0;0;1)
```

Current date

Current date {(*)} -> 戻り値

引数	型		説明
*	演算子	→	サーバの日付を返す
戻り値	日付	↩	現在の日付

説明

Current date コマンドは、システムクロックに保持された日付を現在の日付として返します。

4D Server: リモートモードの4Dでこの関数を実行する際にアスタリスク (*) 引数を渡すと、サーバの現在の日付が返されます。

例題 1

以下の例は、現在の日付を警告ダイアログに表示します。:

```
ALERT("日付は "+String(Current date)+"です。")
```

例題 2

国際市場に向けたアプリケーションを作成する場合に、使用している4Dのバージョンが日付フォーマットMM/DD/YYYY(US版)やDD/MM/YYYY(フランス版)に対応するかどうかを知る必要があるとします。これはデータ入力フィールドのカスタマイズのために知っておくと便利な情報です。

以下のプロジェクトメソッドにより目的の処理を行うことができます:

```
 ` Sys date format プロジェクトメソッド
 ` Sys date format -> String
 ` Sys date format -> デフォルトの4D日付フォーマット

C_STRING(31;$0;$vsDate;$vsMDY;$vsMonth;$vsDay;$vsYear)
C_LONGINT($1;$vlPos)
C_DATE($vdDate)

 ` 月、日、年の値がすべて異なる日付の値を取得
$vdDate:=Current date
REPEAT
   $vsMonth:=String(Month of($vdDate))
   $vsDay:=String(Day of($vdDate))
   $vsYear:=String(Year of($vdDate)%100)
   If(($vsMonth=$vsDay)|($vsMonth=$vsYear)|($vsDay=$vsYear))
      vOK:=0
      $vdDate:=$vdDate+1
   Else
      vOK:=1
   End if
Until(vOK=1)
$0:="" ` 関数の戻り値を初期化
$vsDate:=String($vdDate)
$vlPos:=Position("/";$vsDate) ` 文字列..から最初のセパレータ / を探す
$vsMDY:=Substring($vsDate;1;$vlPos-1) ` 日付から最初の桁を取り出す
$vsDate:=Substring($vsDate;$vlPos+1) ` 最初の区切り文字 / とともに最初の桁を取り除く
Case of
   :($vsMDY=$vsMonth) ` 月を表わす桁
      $0:="MM"
   :($vsMDY=$vsDay) ` 日を表わす桁
      $0:="DD"
   :($vsMDY=$vsYear) ` 年を表わす桁
      $0:="YYYY"
End case
$0:=$0+"/" ` 関数の結果の組み立て開始
$vlPos:=Position("/";$vsDate) ` 文字列../..から2番目のセパレータを探す
$vsMDY:=Substring($vsDate;1;$vlPos-1) ` 日付から以下の桁を取り出す
$vsDate:=Substring($vsDate;$vlPos+1) ` 文字列を日付の最後の桁まで減らす
Case of
   :($vsMDY=$vsMonth) ` 月を表わす桁
      $0:=$0+"MM"
   :($vsMDY=$vsDay) ` 日を表わす桁
      $0:=$0+"DD"
   :($vsMDY=$vsYear) ` 年を表わす桁
      $0:=$0+"YYYY"
End case
$0:=$0+"/" ` 関数結果の構築を続行
Case of
   :($vsDate=$vsMonth) ` 月を表わす桁
      $0:=$0+"MM"
   :($vsDate=$vsDay) ` 日を表わす桁
      $0:=$0+"DD"
   :($vsDate=$vsYear) ` 年を表わす桁
      $0:=$0+"YYYY"
End case
 ` この時点で、$0 はMM/DD/YYYYまたはDD/MM/YYYYまたは...
```

Current time

Current time {(*)} -> 戻り値

引数	型		説明
*	演算子	→	サーバの時刻を返す
戻り値	時間	↩	現在の時刻

説明

Current time コマンドは、システムクロックの現在の時刻を返します。

現在の時刻は常に00:00:00から23:59:59の間です。StringまたはTime stringを使用して、Current timeから返される時間式の文字列を得ることができます。

4D Server: この関数を実行する際に、リモートモードの4Dでアスタリスク (*) 引数を使用すると、サーバの現在時刻が返されます。

例題 1

以下の例題は、処理時間を計測する方法を紹介しています。ここでは、LongOperationメソッドの実行にかかる時間を計っています:

```
$vhStartTime:=Current time ` 開始時刻を保存
LongOperation ` 処理を実行
ALERT("処理時間: "+String(Current time-$vhStartTime)) ` 処理時間を表示
```

例題 2

以下の例題は現在時刻から時間、分、秒を取り出します:

```
$vhNow:=Current time
ALERT("Current hour is: "+String($vhNow\3600))
ALERT("Current minute is: "+String(($vhNow\60)%60))
ALERT("Current second is: "+String($vhNow%60))
```

Date

Date (dateString) -> 戻り値

引数	型		説明
dateString	文字	→	日付を表す文字列
戻り値	日付	⮌	日付

説明

Dateコマンドは、dateString を解釈し、日付を返します。

dateString 引数は、ISO日付フォーマットかシステムレベルで設定される地域設定に従わなければなりません。

ISO日付フォーマット

この文字列は"YYYY-MM-DDTHH:MM:SS"という書式を持った13文字で構成されます。例えば"2013-11-20T10:20:00"と表現され、地域設定と関わり無く、Dateコマンドは引数として渡されたdateStringを評価します。もしdateStringのフォーマットがISO日付フォーマットに適合しない場合は、地域設定で定義された短い形式の日付フォーマットとして評価します。

地域設定

dateStringがISO日付フォーマットに適合しないとき、オペレーティングシステムの地域設定で定義された短い形式の日付フォーマットを評価のために使います。

日本語環境では、日付は通常YY/MM/DD (年, 月, 日) の順です。月と日は1あるいは2桁の数字です。年は2あるいは4桁の数字です。年が2桁の場合は数値に基づいて、19または20を年の先頭に追加します。その境界値は、デフォルトで30です。

- 30以上の場合、20世紀の日付であると判定して、先頭に19を加えます。
- 30未満の場合、21世紀の日付であると判定して、先頭に20を加えます。

SET DEFAULT CENTURYコマンドで境界値を設定できます。

有効な日付区切り文字は、スラッシュ (/), スペース, ピリオド (.), コンマ (,) そしてダッシュ(-)です。

無効な日付("94/13/35" や "94/aa/12" など)を dateString に渡した場合、**Date** はヌル日付を返します(!00/00/00!)。dateString の値が有効であるかを検証するのはデベロッパの責任です。

例題 1

以下の例題はリクエストダイアログを使用して、ユーザに日付の入力を要求します。ユーザが入力した文字列は日付に変換され、reqDate変数に格納されます:

```
reqDate:=Date(Request("日付を入力してください:";String(Current date)))
If(OK=1)
 // reqDate変数に格納された日付に処理を行う
End if
```

例題 2

以下の例題は文字列 "94/12/20"を日付にして返します:

```
vdDate:=Date("94/12/20")
```

例題 3

以下の例題はISOフォーマットに基づき日付に変換します:

```
$vtDateISO:="2013-06-05T20:00:00"
$vDate:=Date($vtDateISO)
 //$vDate変数は、システムに関係なく2013年6月5日を表します
```

Day number

Day number (aDate) -> 戻り値

引数	型	説明
aDate	日付	→ 曜日に対応する数値を得る日付
戻り値	倍長整数	↩ 曜日を示す数値

説明

Day number コマンドは、aDateに対応するの曜日を数値で返します。

Note: Day number は日付がヌルの時、2 を返します。

4Dは "" テーマで以下の定義済み定数を提供します:

定数	型	値
Sunday	倍長整数	1
Monday	倍長整数	2
Tuesday	倍長整数	3
Wednesday	倍長整数	4
Thursday	倍長整数	5
Friday	倍長整数	6
Saturday	倍長整数	7

Note: Day number は1から7までの値を返します。日付から日を取り出すには、Day ofコマンドを使用します。

例題

以下の関数は、現在の曜日文字列を返します。:

```
$viDay :=Day number(Current date) ` $viDay には現在の曜日を示す番号が代入される
Case of
   :($viDay =1)
      $0:="Sunday"
   :($viDay =2)
      $0:="Monday"
   :($viDay =3)
      $0:="Tuesday"
   :($viDay =4)
      $0:="Wednesday"
   :($viDay =5)
      $0:="Thursday"
   :($viDay =6)
      $0:="Friday"
   :($viDay =7)
      $0:="Saturday"
End case
```

Day of

Day of (date) -> 戻り値

引数	型		説明
date	日付	→	日を取り出す日付
戻り値	倍長整数	↩	日を表す数値

説明

Day of コマンドは、dateから日を返します。

Note: Day of は1から31までの値を返します。日付から曜日を取得するには、Day numberコマンドを使用します。

例題 1

以下の例は、Day ofの使用方法を示しています。結果は変数vResultに代入されます。変数vResultに代入される内容はコメントで説明されています:

```
vResult:=Day of(!92/12/25!) // vResult には25が代入されます。
vResult:=Day of(Current date) // vResult 現在の日が代入されます
```

例題 2

Current dateコマンドの例を参照。

Milliseconds

Milliseconds -> 戻り値

引数	型	説明
戻り値	倍長整数	↩ マシンが起動されてからの 経過ミリ秒数

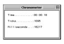
説明

Milliseconds は、マシンが起動されてから経過したミリ秒 (1/1000秒) 数を返します。

例題

以下のコードは1分のクロノメータを表示します:

```
Open window(100;100;300;200;0;"Chronometer")
$vhTimeStart:=Current time
$vlTicksStart:=Tickcount
$vrMillisecondsStart:=Milliseconds
Repeat
   GOTO XY(2;1)
   MESSAGE("時間...........:"+String(Current time-$vhTimeStart))
   GOTO XY(2;3)
   MESSAGE("Tick..........:"+String(Tickcount-$vlTicksStart))
   GOTO XY(2;5)
   MESSAGE("ミリ秒...:"+String(Milliseconds-$vrMillisecondsStart))
Until((Current time-$vhTimeStart)>=?00:01:00?)
CLOSE WINDOW
```



```
Time...........:00:00:18

Ticks..........:1095

Milliseconds...:18217
```

Month of

Month of (aDate) -> 戻り値

引数	型	説明
aDate	日付	→ 月を取り出す日付
戻り値	倍長整数	↩ 日付の月を示す数値

説明

Month of コマンドはaDateの月を返します。

Note: Month of は月の数値を返します。月の名前ではありません (例題 1参照)。

この関数から返される値を比較するために、4DはDays and Monthsテーマに定義済み定数を提供しています:

定数	型	値
January	倍長整数	1
February	倍長整数	2
March	倍長整数	3
April	倍長整数	4
May	倍長整数	5
June	倍長整数	6
July	倍長整数	7
August	倍長整数	8
September	倍長整数	9
October	倍長整数	10
November	倍長整数	11
December	倍長整数	12

例題 1

以下の例題はMonth ofの使用方法を紹介しています。結果は変数vResultに代入されます:

```
vResult:=Month of(!92/12/25!) ` vResult には12が代入される
vResult:=Month of(Current date) ` vResult には今日の日付の月が代入される
```

例題 2

Current dateコマンドの例題を参照。

SET DEFAULT CENTURY

SET DEFAULT CENTURY (century {; pivotYear})

引数	型		説明
century	倍長整数	→	2桁の年数が入力された場合の デフォルト世紀 (マイナス1)
pivotYear	倍長整数	→	2桁の年で日付が入力された時の区切り年

説明

SET DEFAULT CENTURY コマンドは、2桁の年で日付を入力した際に4Dが使用する、デフォルトの世紀と区切り年を指定するために使用します。

区切り年の値は、2桁の年が入力された際に、4Dがどのように日付を解釈するかを定義します:

- 年が区切り年以上の場合、4Dはデフォルト世紀を使用します。
- 年が区切り年未満の場合、4Dは次の世紀を使用します。

デフォルトで、4Dは世紀が20世紀であるとし、区切り年に30を使用します。例えば:

- 97/01/25 は1997年1月25日を意味します。
- 30/01/25 は1930年1月25日を意味します。
- 29/01/25 は2029年1月25日を意味します。
- 07/01/25 は2007年1月25日を意味します。

デフォルトを変更するには、コマンドを実行します。コマンドは即座に効果を持ちます。新しいデフォルト世紀のみ、または新しいデフォルト世紀と区切り年を渡すことができます。

新しいデフォルト世紀マイナス1をcenturyに渡すと、4Dは2桁で入力された日付がこの世紀に属するものとして解釈します。
例えば:

```
SET DEFAULT CENTURY(20) ` 21世紀をデフォルト世紀とする
```

- 97/01/25 は2097年1月25日を意味します。
- 07/01/25 は2007年1月25日を意味します。

さらに、オプションでpivotYear 引数を指定できます。
例えば、以下のように呼び出すと、区切り年は1995になります:

```
SET DEFAULT CENTURY(19;95) ` 年が95未満の場合のデフォルト世紀を21世紀とする
```

- 97/01/25 は1997年1月25日を意味します。
- 07/01/25 は2007年1月25日を意味します。

Note: このコマンドは、2桁の年で日付が入力された際の4Dの日付の解釈にのみ影響します。
以下のようなケースでは:

- 1997/01/25 は1997年1月25日を意味します。
- 2097/01/25 は2097年1月25日を意味します。
- 1907/01/25 は1907年1月25日を意味します。
- 2007/01/25 は2007年1月25日を意味します。

このコマンドはデータ入力にのみ影響します。日付の格納や計算には影響しません。

Tickcount

Tickcount -> 戻り値

引数	型		説明
戻り値	倍長整数	←	マシンが起動されてからの 経過Tick数 (1/60秒)

説明

Tickcount はマシンが起動されてから経過したTick (1/60秒) 数を返します。

Note: Tickcount は倍長整数型の値を返します。

例題

Millisecondsコマンドの例題を参照。

Time

Time (timeValue) -> 戻り値

引数	型		説明
timeValue	文字, 倍長整数	→	時間を表す文字列または倍長整数
戻り値	時間	↩	時間

説明

Time コマンドは、timeValue 引数で指定した時間に相当する時間表示を返します。

timeValue 引数には以下のものを渡す事ができます。

- システムの言語に対応している4Dの標準時間フォーマットで表された時刻を含んでいる文字列(詳細な情報についてはString を参照して下さい)。
- 00:00:00を経過した秒数を表す倍長整数

例題 1

以下の例題は警告ダイアログに"1:00 P.M. = 13 時 0 分"と表示します:

```
ALERT("1:00 P.M. = "+String(Time("13:00:00");Hour Min))
```

例題 2

どんな数字の値も時間として表現することができます。

```
vTime:=Time(10000)
 //vTime は 02:46:40
vTime2:=Time((60*60)+(20*60)+5200)
 //vTime2 は 02:46:40
```

Time string

Time string (seconds) -> 戻り値

引数	型		説明
seconds	倍長整数, 時間	→	0時からの秒数
戻り値	文字	↩	24時フォーマットの時間文字列

説明

Time string コマンドは、seconds に渡した時間式の文字列を返します。

文字列はHH:MM:SS形式です。

1日の秒数は86,400秒ですが、この値を越えても、Time stringは、時間、分、秒の加算を継続します。例えば、Time string (86401) は、文字列"24:00:01"を返します。

Note: 時間を様々なフォーマットの文字列に変換したい場合は、Stringを使用します。

例題

以下の例題は警告ダイアログに"46800 seconds is 13:00:00."というメッセージを表示します。

```
ALERT("46800 seconds is "+Time string(46800))
```

Year of

Year of (date) -> 戻り値

引数	型		説明
date	日付	→	年を取り出す日付
戻り値	倍長整数	←	年を示す数値

説明

Year of コマンドは、dateから年を返します。

例題 1

以下はYear ofの使用例です。結果をvResult変数に代入します。

```
vResult:=Year of(!92/12/25!) ` vResult gets 1992
vResult:=Year of(!1992/12/25!) ` vResult gets 1992
vResult:=Year of(!1892/12/25!) ` vResult gets 1892
vResult:=Year of(!2092/12/25!) ` vResult gets 2092
vResult:=Year of(Current date) ` vResult gets year of current date
```

例題 2

Current dateの例を参照。

暗号化プロトコル

- GENERATE CERTIFICATE REQUEST
- GENERATE ENCRYPTION KEYPAIR

GENERATE CERTIFICATE REQUEST

GENERATE CERTIFICATE REQUEST (privKey ; certifRequest ; codeArray ; nameArray)

引数	型		説明
privKey	BLOB	→	秘密鍵BLOB
certifRequest	BLOB	←	CSRを受け取るBLOB
codeArray	倍長整数配列	→	情報コードリスト
nameArray	文字配列	→	名前リスト

説明

GENERATE CERTIFICATE REQUEST コマンドは、Verisign(R)等の認証局で使用されているPKCSフォーマットで証明書リクエストを生成します。証明書はSSL暗号化プロトコルの重要な役割を持ちます。これはSSLモードで接続している各ブラウザに送信され、Webサイトの"IDカード" (このコマンドに指定した情報をもとに作成) とともに、ブラウザが受信情報の解読に使用できる公開鍵も納められています。さらにこの証明書には、整合性を保証する認証局により加えられた各種情報も納められます。

Note: 4D Webサーバで使用するSSLプロトコルに関する詳細はSSLプロトコルの使用の節を参照してください。

証明書リクエストには、GENERATE ENCRYPTION KEYPAIRコマンドで生成した一対の鍵が使用され、各種情報が納められます。認証局では、このリクエストと他の引数を組み合わせて証明書を作成します。

privKeyにはGENERATE ENCRYPTION KEYPAIRコマンドで生成した秘密鍵を納めたBLOBを渡します。

certifRequestには空のBLOBを渡します。コマンドが実行されると、この引数には証明書リクエストがPKCSフォーマットで納められます。このリクエストを認証局へ提出する目的で、例えばBLOB TO DOCUMENTコマンドを使用して、テキストファイルへ保存することができます。

警告: 秘密鍵はリクエストの作成に使用しますが、認証局へ送信してはいけません。

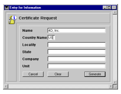
codeArray (倍長整数) およびnameArray (文字列) にはそれぞれ、認証局から要求されるコード番号と情報内容を納めます。
必要とされるコードおよび名称は、認証局や証明書の用途によって変わる場合があります。しかし、証明書の通常の用途であれば (SSL経由でのWebサーバ接続)、この配列には以下の項目を納める必要があります:

提供する情報	codeArray	nameArray (例)
CommonName	13	www.4D-Japan.com
CountryName (two letters)	14	JP
LocalityName	15	Setagaya-ku
StateOrProvinceName	16	Tokyo
OrganizationName	17	4D Japan, Ltd.
OrganizationUnit	18	Web Administrator

コードと情報内容の入力順は問いませんが、これら2つの配列は同期していなくてはなりません。つまり、codeArrayの3番目の項目の値が15 (LocalityName) であれば、nameArrayの3番目の項目にはその情報を納める必要があります。ここの例題ではSetagaya-kuになります。

例題

"Certificate request"フォームには、標準の証明書リクエストで必要となる6つのフィールドが含まれています。**Generate**ボタンは証明書リクエストを納めたドキュメントをディスク上に作成します。(GENERATE ENCRYPTION KEYPAIRコマンドで作成された) 秘密鍵を納めた"Privatekey.txt"もディスク上に存在している必要があります:



Name: 4D, Inc.
Country Name: US

以下は**Generate**ボタンのメソッドです:

```
   ` bGenerate オブジェクトメソッド

C_BLOB($vbprivateKey;$vbcertifRequest)
C_LONGINT($tableNum)
ARRAY LONGINT($tLCodes;6)
ARRAY STRING(80;$tSInfos;6)

$tableNum:=Table(Current form table)
For($i;1;6)
   $tSInfos{$i}:=Field($tableNum;$i)->
   $tLCodes{$i}:=$i+12
End for
If(Find in array($tSInfos;"")#-1)
   ALERT("All fields should be filled.")
Else
   ALERT("Select your private key.")
   $vhDocRef:=Open document("")
   If(OK=1)
      CLOSE DOCUMENT($vhDocRef)
      DOCUMENT TO BLOB(Document;$vbprivateKey)
      GENERATE CERTIFICATE REQUEST($vbPrivateKey;$vbcertifRequest;$tLCodes;$tSInfos)
      BLOB TO DOCUMENT("Request.txt";$vbcertifRequest)
   Else
      ALERT("Invalid private key.")
   End if
End if
```

GENERATE ENCRYPTION KEYPAIR

GENERATE ENCRYPTION KEYPAIR (privKey ; pubKey {; length})

引数	型		説明
privKey	BLOB	←	秘密鍵を受け取るBLOB
pubKey	BLOB	←	公開鍵を受け取るBLOB
length	倍長整数	→	キー長 (ビット) [386...2048] デフォルト値 = 512

説明

GENERATE ENCRYPTION KEYPAIR コマンドは新しく1対のRSA鍵を生成します。4Dで提供されるセキュリティシステムは、情報の暗号/復号のために設計されたこれらの鍵をベースにしています。これらの鍵は4D Webサーバ (暗号化および暗号化通信) のSSLプロトコル、およびすべてのデータベース (データの暗号化) で使用できます。

コマンドが実行されると、privKeyとpubKeyに渡されたBLOBには新しい1対の暗号鍵が納められます。

オプションの引数lengthを使用して鍵のサイズ (ビット単位) を設定することができます。鍵が大きいほど暗号コードの解読は困難になります。

ただし鍵が大きくなると実行時間や応答時間が長くなり、特にSSL接続ではこれが顕著です。

デフォルトでは (lengthを省略した場合)、生成される鍵のサイズは512ビットに設定されます。暗号の安全性を高めるために頻繁に、例えば6か月ごとに鍵を交換することもできます。2,048ビットの鍵を生成できますが、Webアプリケーションの接続速度は低下します。

このコマンドはPKCSフォーマットで鍵を生成し、その内容を変更する必要なく電子メールにコピー＆ペーストできます。一対の鍵が生成されたら (例えばBLOB TO DOCUMENTコマンドを使用して) テキストドキュメントを作成することができ、これらの鍵を安全な場所に保管できます。

警告: 秘密鍵は常に秘密にしなければなりません。

RSA、秘密鍵、および公開鍵について

GENERATE ENCRYPTION KEYPAIRで使用するRSA暗号方 式は、秘密鍵と公開鍵という二重鍵暗号システムに基づいています。その名が示す通り、公開鍵は第三者に渡され、情報の復号に使用されます。公開鍵は情報の暗号化に使われるユニークな秘密鍵と一対です。このように、秘密鍵は暗号化に使用され、公開鍵は復号に使用されます (またはその逆)。一方の鍵を使って暗号化された情報は、もう一方の鍵を使用しなければ解読することはできません。

SSLプロトコルの暗号化機能はこの原理に基づいており、証明書に納められた公開鍵がブラウザに送信されます。(詳細はSSLプロトコルの使用の節を参照)。

この暗号化モードは、ENCRYPT BLOBおよびDECRYPT BLOBの1番目のシンタックスでも使用されています。このシンタックスで用いる公開鍵は極秘に発行してください。
特定の受信者を復号可能な唯一の人とし、かつ送信者が暗号化を行った人であることを保証するために、2人の公開鍵と秘密鍵を合わせて情報の暗号化を行うことができます。この原理はENCRYPT BLOBおよびDECRYPT BLOBの2番目のシンタックスで示されています。

例題

ENCRYPT BLOBの例題参照

演算子

- 演算子
- ビットワイズ演算子
- 比較演算子
- 日付演算子
- 論理演算子
- 数値演算子
- ピクチャ演算子
- 文字列演算子
- 時間演算子

演算子

演算子とは、式の間で行われる操作を指定するために使用する記号です。演算子の機能は、以下のとおりです:

- 数値、日付、時間の計算を行います。
- 文字列操作、論理式における論理演算、図形上での特殊演算を行います。
- 単純な式を組み合わせて新たな式を生成します。

優先順位

式を評価する順番を優先順位と呼びます。4Dの優先順位は厳密に左から右で、代数的順序は採用されていません。次の例を見てみましょう:

```
3+4*5
```

これは35を返します。最初に式3+4の結果7を求め、それに5を乗じるので、結果は35になります。

左から右の優先順位を変更するには、必ずカッコを使用します。例えば:

```
3+(4*5)
```

この式は、23を返します。カッコがあるため、最初に式（4*5）の結果20を求め、それに3を加えて、結果は23になります。

カッコは、他のカッコの組の内側にネストすることができます。式の評価が正しく行われるように、必ず各左カッコに対応する右カッコを指定してください。カッコが足りなかったり、誤って使用した場合は、予測できない結果、または式が無効になります。またコンパイル行おうとする場合は、左カッコと右カッコは同じ数でなければなりません。コンパイラは、組になっていないカッコをシンタックスエラーとして検出します。

代入演算子

代入演算 := は必ず他の演算と区別しなければなりません。代入演算子は、式を組み合わせて新しい一つのものにするのではなく、右側の式の値を演算子の左側の変数やフィールドにコピーします。例えば、次のステートメントは数値4（"Acme"の文字数）を変数MyVarに代入します。MyVarのデータタイプは数値です。

```
MyVar:=Length("Acme")
```

重要: 代入演算子:=と等号比較演算子=とを混同しないでください。

4D言語の他の演算子については、以下の節で説明します:

文字列演算子

の節を参照してください。

数値演算子

の節を参照してください。

日付演算子

の節を参照してください。

時間演算子

の節を参照してください。

比較演算子

の節を参照してください。

論理演算子

の節を参照してください。

ピクチャ演算子

の節を参照してください。

ビットワイズ演算子

の節を参照してください。

ビットワイズ演算子

ビットワイズ演算子は、倍長整数式や値に対して演算を行います。

Note: ビットワイズ演算子に整数値または実数値を渡すと、4Dは値を倍長整数値として評価してから、ビットワイズ演算子を使用した式を計算します。

ビットワイズ演算子を使用する場合、倍長整数値を32ビットの配列と考える必要があります。これらのビットには、右から左に0~31の番号が付けられます。

それぞれのビットは0か1なので、倍長整数値は32の論理値を格納できる値と考えることもできます。1に等しいビットはTrue、0に等しいビットはFalseを意味します。

ビットワイズ演算子を使用する式は倍長整数値を返します。Bit Test演算子は例外的に式がブール値を返します。次の表にビットワイズ演算子とそのシンタックスを示します:

処理	**演算子**	**シンタックス**	**戻り値**	
Bitwise AND	&	Long & Long	Long	
Bitwise OR (inclusive)	|	Long | Long	Long	
Bitwise XOR (exclusive)	^|	Long ^| Long	Long	
Left Bit Shift	<<	Long << Long	Long	(see note 1)
Right Bit Shift	>>	Long >> Long	Long	(see note 1)
Bit Set	?+	Long ?+ Long	Long	(see note 2)
Bit Clear	?-	Long ?- Long	Long	(see note 2)
Bit Test	??	Long ?? Long	Bool	(see note 2)

Notes

(1) **Left Bit Shift**および**Right Bit Shift**演算では、2番目のオペランドは、結果値において1番目のオペランドのビットがシフトされるビット数を示します。したがって、この2番目のオペランドは、0~32の間でなければなりません。0ビットシフトするとその値がそのまま返されます。また、31ビットより多くシフトするとすべてのビットがなくなるので、0x00000000が返されます。それ以外の値を2番目のオペランドとして渡した場合、結果は意味のない値になります。
(2) **Bit Set**、**Bit Clear**、**Bit Test**演算では、2番目のオペランドは、作用の対象となるビット番号を示します。したがって、この2番目のオペランドは0~31の間です。そうでない場合、**Bit Set**と**Bit Clear**に対しては1番目のオペランドがそのまま返され、**Bit Test**に対してはFalseが返されます。

次の表は、ビットワイズ演算子とその効果を示します:

処理	**説明**
Bitwise AND	それぞれの結果ビットは2つのオペランドのビットの論理ANDです。 下記は、論理ANDの真偽表です: 1 & 1 --> 1 0 & 1 --> 0 1 & 0 --> 0 0 & 0 --> 0 すなわち、両オペランドのビットが1の場合、結果ビットが1になり、その他の場合は結果ビットが0になります。
Bitwise OR (inclusive)	それぞれの結果ビットは2つのオペランドのビットの論理ORです。 下記は、論理ORの真偽表です: 1 | 1 --> 1 0 | 1 --> 1 1 | 0 --> 1 0 | 0 --> 0 すなわち、いずれかのオペランドのビットが1の場合、 結果ビットが1になり、その他の場合は結果ビットが0になります。
Bitwise XOR (exclusive)	それぞれの結果ビットは2つのオペランドのビットの排他的論理ORです。 下記は、排他的論理ORの真偽表です: 1 ^| 1 --> 0 0 ^| 1 --> 1 1 ^| 0 --> 1 0 ^| 0 --> 0 すなわち、オペランドのビットのいずれか一方だけが1の場合、 結果ビットが1になり、その他の場合は結果ビットが0になります。
Left Bit Shift	最初のオペランド値が結果値に設定され、次に結果ビットが2番目のオペランドで示されたビット数だけ左にシフトします。左側のビットがなくなり、右側の新しいビットは0に設定されます。 Note: 正の数だけを考えると、Nビット左にシフトすることは、2^Nを掛けることと同じです。
Right Bit Shift	最初のオペランド値が結果値に設定され、次に結果ビットが2番目のオペランドで示されたビット数だけ右にシフトします。右側のビットがなくなり、左側の新しいビットは0に設定されます。 Note: 正の数だけを考えると、Nビット左にシフトすることは、2^Nで割ることと同じです。
Bit Set	最初のオペランド値が結果値に設定され、次に結果ビットのうち2番目のオペランドで示されたビットが1に設定されます。他のビットはそのままです。
Bit Clear	最初のオペランド値が結果値に設定され、次に結果ビットのうち2番目のオペランドで示されたビットが0に設定されます。他のビットはそのままです。
Bit Test	最初のオペランドのうち、2番目のビットで示されたビットが1の場合、Trueが返されます。最初のオペランドのうち、2番目のビットで示されたビットが0の場合、Falseが返されます。

例題

(1) 次の表にそれぞれのビット演算子の例を示します:

処理	**例題**	**結果**
Bitwise AND	0x0000FFFF & 0xFF00FF00	0x0000FF00
Bitwise OR (inclusive)	0x0000FFFF | 0xFF00FF00	0xFF00FFFF
Bitwise XOR (exclusive)	0x0000FFFF ^| 0xFF00FF00	0xFF0000FF
Left Bit Shift	0x0000FFFF << 8	0x00FFFF00
Right Bit Shift	0x0000FFFF >> 8	0x000000FF
Bit Set	0x00000000 ?+ 16	0x00010000
Bit Clear	0x00010000 ?- 16	0x00000000
Bit Test	0x00010000 ?? 16	True

(2) 4Dは多くの定義済み定数を提供しています。これらの定数の中には、そのリテラルが"bit"または"mask"で終わるものがあります。例えばテーマで提供される定数の場合を以下に示します:

定数	**型**	**値**
Changed resource bit	倍長整数	1
Changed resource mask	倍長整数	2
Locked resource bit	倍長整数	4
Locked resource mask	倍長整数	16
Preloaded resource bit	倍長整数	2
Preloaded resource mask	倍長整数	4
Protected resource bit	倍長整数	3
Protected resource mask	倍長整数	8
Purgeable resource bit	倍長整数	5
Purgeable resource mask	倍長整数	32
System heap resource bit	倍長整数	6
System heap resource mask	倍長整数	64

これらの定数により、Get resource propertiesが返した値をテストしたり、SET RESOURCE PROPERTIESに渡す値を作成したりすることができます。リテラルが"bit"で終わる定数は、テスト、消去またはセットするビットの位置を指定します。リテラルが"mask"で終わる定数は、テスト、消去またはセットするビットが1の場合のみ、倍長整数値を指定します。

例えば、(変数$vlResAttrにプロパティを取得した) リソースがパージ可能かどうかをテストするには、以下のように記述します:

```
If($vlResAttr ?? Purgeable resource bit)  ` リソースはパージ可能か?
```

または:

```
If(($vlResAttr  & Purgeable resource mask)#0)  ` リソースはパージ可能か?
```

逆に、これらの定数を使用して同じビットを設定することもできます。以下のように記述します:

```
$vlResAttr:=$vlResAttr ?+Purgeable resource bit
```

または:

```
$vlResAttr:=$vlResAttr |Purgeable resource bit
```

(3) この例では、2つの整数値を倍長整数値に格納します。以下のように記述します:

```
$vlLong:=($viIntA<<16)|$viIntB  ` 2つの整数を倍長整数に格納
```

```
$vlIntA:=$vlLong>>16  ` ハイワードに格納されている整数を取り出します
```

```
$viIntB:=$vlLong  & 0xFFFF  ` ローワードに格納されている整数を取り出します
```

Tip: 倍長整数または整数の値を数値とビットワイズ演算子を組み合わせた式で処理する場合は注意が必要です。ハイビット(倍長整数の場合はビット31、整数の場合はビット15)は符号を設定します。その値が消去された場合には正の数で、値が設定された場合には負の数です。数値演算子はこのビットを使用して、値の符号を検出します。ビットワイズ演算子はこのビットの意味を無視します。

比較演算子

この節の表は文字列、数値、日付、時間、ポインターおよびメタデータ式を持つピクチャー式に適用する比較演算子を示しています。配列やBLOBには使用できません。比較演算子を使用する式はTRUEまたはFALSEのブール値を返します。

注: 2つのピクチャーをEqual picturesを使用して比較できます。

文字列比較

演算子	シンタックス	戻り値	例	結果
= (等しい)	文字列 = 文字列	ブール	"abc" = "abc"	True
			"abc" = "abd"	False
# (異なる)	文字列 # 文字列	ブール	"abc" # "abd"	True
			"abc" # "abc"	False
> (大きい)	文字列 > 文字列	ブール	"abd" > "abc"	True
			"abc" > "abc"	False
< (小さい)	文字列 < 文字列	ブール	"abc" < "abd"	True
			"abc" < "abc"	False
>= (以上)	文字列 >= 文字列	ブール	"abd" >= "abc"	True
			"abc" >= "abd"	False
<= (以下)	文字列 <= 文字列	ブール	"abc" <= "abd"	True
			"abd" <= "abc"	False
% (キーワードを含む)	文字列 % 文字列	ブール	"Alpha Bravo" % "Bravo"	True
			"Alpha Bravo" % "ravo"	False
	Picture % String	Boolean	*Picture_expr* % "Mer"	True (*)

(*) キーワード"Mer"がピクチャー式 (フィールドまたは変数) に格納されたピクチャーに割り当てられていた場合。

重要: この節の終りに、文字列比較に関する情報が提供されています。

数値比較

演算子	シンタックス	戻り値	例	結果
= (等しい)	数値 = 数値	ブール	10 = 10	True
			10 = 11	False
# (異なる)	数値 # 数値	ブール	10 #11	True
			10 # 10	False
> (大きい)	数値 > 数値	ブール	11 > 10	True
			10 > 11	False
< (小さい)	数値 < 数値	ブール	10 < 11	True
			11 < 10	False
>= (以上)	数値 >= 数値	ブール	11 >= 10	True
			10 >= 11	False
<= (以下)	数値 <= 数値	ブール	10 <= 11	True
			11 <= 10	False

日付比較

演算子	シンタックス	戻り値	例	結果
= (等しい)	日付 = 日付	ブール	!1/1/97! =!1/1/97!	True
			!1/20/97! =!1/1/97!	False
# (異なる)	日付 # 日付	ブール	!1/20/97! # !1/1/97!	True
			!1/1/97! # !1/1/97!	False
> (大きい)	日付 > 日付	ブール	!1/20/97! > !1/1/97!	True
			!1/1/97! > !1/1/97!	False
< (小さい)	日付 < 日付	ブール	!1/1/97! < !1/20/97!	True
			!1/1/97! < !1/1/97!	False
>= (以上)	日付 >= 日付	ブール	!1/20/97! >=!1/1/97!	True
			!1/1/97!>=!1/20/97!	False
<= (以下)	日付 <= 日付	ブール	!1/1/97!<=!1/20/97!	True
			!1/20/97!<=!1/1/97!	False

時間比較

演算子	シンタックス	戻り値	例	結果
= (等しい)	時間 = 時間	ブール	?01:02:03? = ?01:02:03?	True
			?01:02:03? = ?01:02:04?	False
# (異なる)	時間 # 時間	ブール	?01:02:03? # ?01:02:04?	True
			?01:02:03? # ?01:02:03?	False
> (大きい)	時間 > 時間	ブール	?01:02:04? > ?01:02:03?	True
			?01:02:03? > ?01:02:03?	False
< (小さい)	時間 < 時間	ブール	?01:02:03? < ?01:02:04?	True
			?01:02:03? < ?01:02:03?	False
>= (以上)	時間 >= 時間	ブール	?01:02:03? >=?01:02:03?	True
			?01:02:03? >=?01:02:04?	False
<= (以下)	時間 <= 時間	ブール	?01:02:03? <=?01:02:03?	True
			?01:02:04? <=?01:02:03?	False

ポインタ比較

表中の例は、以下のようにポインターが設定されているものとして表記しています:

```
  ` vPtrA と vPtrB は同じオブジェクトを指します
 vPtrA:=->anObject
 vPtrB:=->anObject
  ` vPtrC は他のオブジェクトを指します
 vPtrC:=->anotherObject
```

演算子	シンタックス	戻り値	例	結果
= (等しい)	Pointer = Pointer	ブール	vPtrA = vPtrB	True
			vPtrA = vPtrC	False
# (異なる)	Pointer # Pointer	ブール	vPtrA # vPtrC	True
			vPtrA # vPtrB	False

文字列比較の詳細

- 文字列は文字ごとに比較されます (後述のキーワードによる検索の場合を除きます)。
- 文字列が比較されるとき文字の大小文字は無視されます。したがって、"a"="A"はTRUEを返します。大文字と小文字を区別して比較するには、文字コードで比較してください。例えば次の式はFALSEです:

  ```
  Character code("A")=Character code("a") ` because 65 is not equal to 97
  ```

- 文字列が比較される場合、文字は使用するコンピュータのシステム文字比較テーブルを使用して比較されます。例えば、以下の式は日本語システムではTRUEを返します:

  ```
  "あ"="ア"
  "ア"="ｱ"
  "１"="1"
  ```

- 他の文字列比較と異なり、キーワードによる検索はテキスト中の単語を検索します: 単語は個々にかつそれが全体として扱われます。複数の単語を検索したり、音節など単語の一部を検索するような場合、% 演算子は常にFalseを返します。単語の判別はICUライブラリを使用して行われます。"Today's"のようにアポストロフィを含む単語は、それを 含め1つの単語として扱われます。数字も検索できます。小数点も数字の一部、すなわち単語として扱われます。ただし通貨記号や温度記号などの記号は無視されます。

  注: データベースの文字列比較の言語が日本語に設定されている場合、キーワードの分割は区切り文字を使用して判定されます。この場合の区切り文字は以下の通りです:
    - 文字コード32 (0x20) 以下の文字
    - ICUの関数u_isalphaとu_isdigitがともにFalseを返す文字

  ```
  "Alpha Bravo Charlie"%"Bravo" ` Returns True
  "Alpha Bravo Charlie"%"vo" ` Returns False
  "Alpha Bravo Charlie"%"Alpha Bravo" ` Returns False
  "Alpha,Bravo,Charlie"%"Alpha" ` Returns True
  "Software and Computers"%"comput@" ` Returns True
  ```

注: キーワードがどのように考慮されるかに関する詳細な情報は、以下のアドレスを参照してください:
http://www.unicode.org/unicode/reports/tr29/#Word_Boundaries.

- ワイルドカード記号 (@) は、すべての文字列の比較に使用することができます。これは任意の数の文字を比較します。したがって、次の式はTRUEになります:

  ```
  "abcdefghij"="abc@"
  ```

  任意の数の文字を比較する目的のワイルドカード記号は、2番目のオペランド (比較演算子の右側の式) で使用しなければなりません。最初のオペランドでは"@"は単なる1文字であると解釈されるため、次の式はFALSEです:

  ```
  "abc@"="abcdefghij"
  ```

  ワイルドカードは "0文字以上"を意味します。以下の式はTRUEです:

  ```
  "abcdefghij"="abcdefghij@"
  "abcdefghij"="@abcdefghij"
  "abcdefghij"="abcd@efghij"
  "abcdefghij"="@abcdefghij@"
  "abcdefghij"="@abcde@fghij@"
  ```

  一方、どのような場合でも、ワイルドカードを2つ連続して使用した文字列比較は常にFALSEを返します。次の式はFALSEになります:

  ```
  "abcdefghij"="abc@@fg"
  ```

  比較演算子が < または > 記号である、あるいはこれらを含む場合、第2オペランドの終りに置かれた1つのワイルドカードのみがサポートされます:

  ```
  "abcd"<="abc@" `有効な比較
  "abcd"<="abc@ef" `有効でない比較
  ```

Tip文字列の比較または検索において、@をワイルドカードではなく一般の文字として扱いたい場合、2つの方法があります:

- Character code (At sign) 指示を使用する

  例えば文字列が @ 文字で終わっているかどうかを知りたいとします。

  - 以下の式は ($vsValue が空でなければ) 常にTRUEです:

  ```
  ($vsValue [[Length($vsValue)]]="@")
  ```

  - 以下の式は正しく評価されます:

  ```
  (Character code($vsValue[[Length($vsValue)]])#64)
  ```

- データベース設定ダイアログボックスの「@がテキストパターンの最初または最後にある場合のみ、ワイルドカードとして扱う」をチェックする

  このオプションは、文字列に@が含まれているとき、@文字がどう解釈されるかを定義します。すなわちクエリと並び替えで、@をどのように扱うかに影響を与えます。詳しくは、4D Design Referenceマニュアルを参照してください。

日付演算子

日付演算子を使用する式は、その処理に応じて日付または数値を返します。日付演算はすべて、年の移動や閏年も考慮に入れて正確な結果を返します。次の表に、日付演算子を示します:

演算子	シンタックス	戻り値	例	結果
日付の差	日付 - 日付	数値	!97/1/20! - !97/1/1!	19
日付の加算	日付 + 数値	日付	!97/1/20! + 9	!97/1/29!
日付の減算	日付 - 数値	日付	!97/1/20! - 9	!97/1/11!

論理演算子

4Dは、論理積 (&) と論理和 (|) をサポートしています。演算子は、両方ともブール式に関して機能します。論理積 (&) は両方の式がTRUEである場合にTRUEを返します。論理和 (|) は少なくとも一方の式がTRUEの時にTRUEを返します。

4Dには True、False、Not**というブール関数もあります。この関数の詳細は、該当する章を参照してください。**

次の表に、論理演算子を示します:

演算子	シンタックス	**戻り値**	**例**	**結果**
AND	ブール & ブール	ブール	("A" = "A") & (15 # 3)	True
			("A" = "B") & (15 # 3)	False
			("A" = "B") & (15 = 3)	False
OR	ブール | ブール	ブール	("A" = "A") | (15 # 3)	True
			("A" = "B") | (15 # 3)	True
			("A" = "B") | (15 = 3)	False

次の表に、論理演算子 (&) の真偽表を示します:

Expr1	Expr2	Expr1 & **Expr2**
True	True	True
True	False	False
False	True	False
False	False	False

次の表に、論理演算子 (|) の真偽表を示します:

Expr1	Expr2	Expr1 | Expr2
True	True	True
True	False	True
False	True	True
False	False	False

Tip式1と式2の排他的結合子演算を実行する必要がある場合、次の評価式を使用します:

```
(Expr1|Expr2) & Not(Expr1 & Expr2)
```

数値演算子

数値演算子を使用する式は数値を返します。次の表に、数値演算子を示します:

演算子	シンタックス	戻り値	例	結果
加算	数値 + 数値	数値	2 + 3	5
減算	数値 - 数値	数値	3 - 2	1
乗算	数値 * 数値	数値	5 * 2	10
除算	数値 /数値	数値	5 / 2	2.5
倍長整数を返す除算	数値 \ 数値	数値	5 \ 2	2
モジューロ	数値 % 数値	数値	5 % 2	1
指数	数値 ^ 数値	数値	2 ^ 3	8

モジューロ演算子モジューロ演算子 (%) は最初の数値を2番目の数値で除算し、その余りの整数を返します。次に例を示します:

- 10 % 2は、0を返します。10は2で割り切れるからです。
- 10 % 3は、1を返します。余りが1だからです。
- 10.5 % 2は、0を返します。余りが整数ではない(0.25)からです。

警告:

- モジューロ演算子 (%) は倍長整数の範囲内(-2^31から(2^31)-1まで)の数値に対して有効値を返します。この範囲外の数値のモジューロ演算を実行するには、Mod コマンドを使用します。
- 倍長整数を返す除算演算子 (\) は、整数値の有効値を返します。

ピクチャ演算子

ピクチャ演算子を使用する式はピクチャを返します。次の表はピクチャ演算子を示ししたものです。

演算子	シンタックス	動作
+ (水平連結)	Pict1 + Pict2	Pict1の右側にPict2を追加する。
/ (垂直連結)	Pict1 / Pict2	Pict1の下側にPict2を追加する。
& (排他的論理和)(*)	Pict1 & Pict2	Pict1の前面にPict2を重ね合わせます(Pict2が前面に来ます)。
| (包括的論理和)(*)	Pict1 | Pict2	Pict1とPict2を重ね、そのマスクした結果を返します(両ピクチャとも同じサイズである必要があります)。
+ (水平移動)	ピクチャ + 数値	ピクチャを指定ピクセル分、横に移動する。
/ (垂直移動)	ピクチャ / 数値	ピクチャを指定ピクセル分、縦に移動する。
* (サイズ変更)	ピクチャ * 数値	割合によってピクチャのサイズを変更する。
*+ (水平スケール)	ピクチャ *+ 数値	割合によって水平にピクチャサイズを変更する。
*/ (垂直スケール)	ピクチャ */ 数値	割合によって垂直にピクチャサイズを変更する。

(*) 排他的論理和 (&) と包括的論理和 (|) の演算子の機能は、プログラムが使用するディスプレイ管理ライブラリのアップデートに伴い、4D v14以降変更されています。
Pict3 := Pict1 & Pict2 は以下と同様の結果の結果になります:

```
COMBINE PICTURES(pict3;pict1;Superimposition;pict2)
```

Pict3 := Pict1 | Pict2 は以下と同様の結果になります:

```
Sequal:=Equal pictures(Pict1;Pict2;Pict3)
```

| 演算子を使用するためには、Pict1 と Pict2 が完全に同一のサイズでなければならないことに注意して下さい。二つのピクチャのサイズに違いがある場合、Pict1 | Pict2 は空のピクチャを生成します。

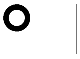
Note: COMBINE PICTURES コマンドは、それぞれのソースピクチャの特性を結果ピクチャに保持しつつ、ピクチャの重ね合わせを行います。

他のピクチャ演算子は、2つの元ピクチャがベクトルの場合に、ベクトルピクチャを返します。しかし、表示形式On Backgroundでプリントされたピクチャはビットマップとしてプリントされる点に留意してください。

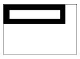
例題

以下の例では、すべてのピクチャが表示フォーマットにOn Backgroundを指定しています。

これはcircleピクチャです:



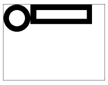
これはrectangleピクチャです:



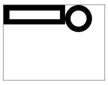
以下の例は、各ピクチャ演算子の効果をグラフィック表現したものです。

水平結合

`circle+rectangle` ` `circle`の右側に`rectangle`が置かれます。

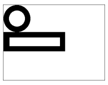


`rectangle+circle` ` `rectangle`の右側に`circle`が置かれます。

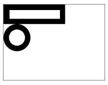


垂直結合

`circle/rectangle` ` `circle`の下側に`rectangle`が置かれます。

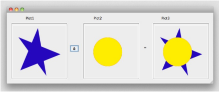


`rectangle/circle` ` `rectangle`の下側に`circle`が置かれます。



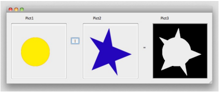
排他的論理和

`Pict3:=Pict1 & Pict2 // Pict1`の上に`Pict2`を重ね合わせます。



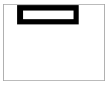
包括的論理和

`Pict3:=Pict1|Pict2 //` 同じサイズの二つのピクチャを重ね合わせた上でそのマスクの結果を返します。

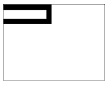


水平移動

`rectangle+50` ` `rectangle`を右に`50`ピクセル移動します。

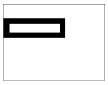


`rectangle-50` ` `rectangle`を左に`50`ピクセル移動します。

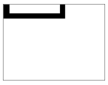


垂直移動

`rectangle/50` ` `rectangle`を下に`50`ピクセル移動します。

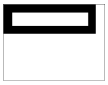


`rectangle/-20` ` `rectangle`を上に`20`ピクセル移動します。

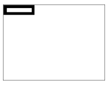


サイズ変更

`rectangle*1.5` ` `rectangle`を`50%`拡大します。

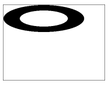


`rectangle*0.5` ` `rectangle`を`50%`縮小します。



水平スケール

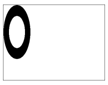
`circle*+3` ` `circle`を`3`倍、水平方向に拡げます。



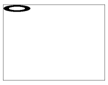
`circle*+0.25` ` `circle`の幅を`25%`にします。

垂直スケール

`circle*/2` ` `circle`を`2`倍の高さにする。



`circle*/0.25` ` `circle`を縦に`25%`にする。



文字列演算子

文字列演算子を使用する式は、文字列を返します。次の表に、文字列演算子を示します:

演算子	シンタックス	戻り値	例	結果
Concatenation	文字列 + 文字列	文字列	"abc" + "def"	"abcdef"
Repetition	文字列 * 数値	文字列	"ab" * 3	"ababab"

時間演算子

時間演算子を使用する式は、その処理に応じて時間または数値を返します。次の表に、時間演算子を示します:

演算子	シンタックス	戻り値	例	結果
加算	時間 + 時間	時間	?02:03:04? + ?01:02:03?	?03:05:07?
減算	時間 - 時間	時間	?02:03:04? - ?01:02:03?	?01:01:01?
加算	時間 + 数値	数値	?02:03:04? + 65	7449
減算	時間 - 数値	数値	?02:03:04? - 65	7319
乗算	時間 * 数値	数値	?02:03:04? * 2	14768
除算	時間 / 数値	数値	?02:03:04? / 2	3692
倍長整数の除算	時間 \ 数値	数値	?02:03:04? \ 2	3692
モジューロ	時間 % 数値	数値	?02:03:04? % 2	0

例題 1

時間式を数値と組み合わせた式から時間式を取得するには、Time コマンドと Time string コマンドを使用します。

Time または Current time ファンクションを使用することによって時間型と数値型の式を組み合わせることが出来ます。例えば:

```
 // 以下の行は $vlSeconds に、深夜0時から現在より1時間後までに経過した秒数を代入します。
$vlSeconds:=Current time+3600
 // 以下の行は $vHSoon 1時間後の時刻を代入します。
$vhSoon:=Time(Current time+3600)
```

2番目の行はより簡単に記述することができます:

```
 // 以下の行は $vHSoon 1時間後の時刻を代入します。
$vhSoon:=Current time+?01:00:00?
```

例題 2

状況によっては、時間表現を数値表現に変換する必要があります。
例えば、Open documentを使用してドキュメントを開くとドキュメント参照 (DocRef) が返されますが、これは元々時間式です。後でドキュメント参照として数値を受け取る4D ExtensionルーチンにDocRefを渡すとします。この場合ゼロに加えることにより、時間値からその値を変更することなく数値を取得できます。

例題:

```
 ` ドキュメントを選択し開く
$vhDocRef:=Open document("")
If(OK=1)
 ` ドキュメント参照値の時間を数値としてルーチンに渡します
   DO SOMETHING SPECIAL(0+$vhDocRef)
End if
```

例題 3

モジューロ演算子を使用できます。とくに24時間フォーマットを考慮した時間の追加に便利です:

```
$t1:=?23:00:00? // これは午後11:00です
 // 2時間30分を追加します
$t2:=$t1 +?02:30:00? // 単純な追加を行うと$t2は?25:30:00?になります。
$t2:=($t1 +?02:30:00?)%?24:00:00? // $t2は?01:30:00?となる。
```

算術関数

- Abs
- Arctan
- Cos
- Dec
- Euro converter
- Exp
- Int
- Log
- Mod
- Random
- Round
- SET REAL COMPARISON LEVEL
- Sin
- Square root
- Tan
- Trunc

Abs

Abs (number) -> 戻り値

引数	型	説明
number	実数	→ 絶対値を求める数値
戻り値	実数	↩ 絶対値

説明

Abs はnumberの絶対値（符号なしの正の値）を返します。numberが負の場合、正の数を返します。numberが正の数の場合は、値は変わりません。

例題

以下の例は、-10.3の絶対値である10.3を返します:

```
vlVector:=Abs(-10.3)
```

Arctan

Arctan (number) -> 戻り値

引数	型		説明
number	実数	→	角度を求めるタンジェント値
戻り値	実数	←	ラジアン角度

説明

Arctan はnumberの逆正接値をラジアンで返します。numberはタンジェントです。

Note: 4DではPi、Degree、Radianという定数があらかじめ定義されています。Piはパイの数値 (3.14159...) を返し、Degreeはラジアンで表わされた1度 (0.01745...) を、Radianは度数で表わされた1ラジアン (57.29577...) を返します。

例題

以下の例はPiの値を表示します:

```
ALERT("Pi is equal to: "+String(Arctan(1)*4))
```

Cos

Cos (number) -> 戻り値

引数	型		説明
number	実数	→	コサインを求めるラジアン値
戻り値	実数	←	コサイン値

説明

Cos はnumberの余弦値を返します。numberはラジアンで指定します。

Note: 4DではPi、Degree、Radianという定数があらかじめ定義されています。Piはパイの数値 (3.14159...) を返し、Degreeはラジアンで表わされた1度 (0.01745...) を、Radianは度数で表わされた1ラジアン (57.29577...) を返します。

Dec

Dec (number) -> 戻り値

引数	型		説明
number	実数	→	小数部を求める数値
戻り値	実数	⮐	小数部の数値

説明

Decはnumberの小数部を返します。返す値は、常に正の数またはゼロになります。

例題

以下の例は、通貨のドル単位で実数になっている金額を、整数のドルとセントに変換します。金額が7.31ドルの場合、7ドルと31セントになります:

```
vlDollars:=Int(vrAmount) ` Get the dollars
vlCents:=Dec(vrAmount)*100 ` Get the fractional part
```

Euro converter

Euro converter (value ; fromCurrency ; toCurrency) -> 戻り値

引数	型		説明
value	実数	⇒	変換する値
fromCurrency	文字	⇒	valueの通貨コード
toCurrency	文字	⇒	変換先通貨コード
戻り値	実数	↩	変換された値

説明

コマンドは"ユーロ"に所属するユーロ通貨の元と先の異なった通貨の値を変換します。

変換できるものは:

- 各国通貨からユーロ
- ユーロから各国通貨
- 各国通貨から他の各国通貨。この場合変換はユーロを仲介して計算されます。例えば、ベルギーフランをドイツマルクに変換すると、4Dは以下の計算を実行します:

ベルギーフラン -> ユーロ -> ドイツマルク

最初の引数を変換する値とします。
2番目の引数は第一引数の通貨コードを示します。
3番目の引数は変換後の通貨コードを示します。

通貨コードを指定するために、4Dは "" の定義済み定数を提供します:

定数	型	値
Austrian Schilling	文字列	ATS
Belgian Franc	文字列	BEF
Deutschemark	文字列	DEM
Euro	文字列	EUR
Finnish Markka	文字列	FIM
French Franc	文字列	FRF
Greek Drachma	文字列	GRD
Irish Pound	文字列	IEP
Italian Lira	文字列	ITL
Luxembourg Franc	文字列	LUF
Netherlands Guilder	文字列	NLG
Portuguese Escudo	文字列	PTE
Spanish Peseta	文字列	ESP

必要な場合、4Dは変換した結果が小数点2位となるよう自動的に四捨五入します。例外としてイタリアリラ、ベルギーフラン、ルクセンブルグフラン、スペインペセタへの変換時、4Dは結果が整数値となるようにします。

ユーロと11の参加メンバー国の通貨の変換レートは固定されています:

通貨	1ユーロの値
Austrian Schilling	13.7603
Belgian Franc	40.3399
Deutschemark	1.95583
Finnish Markka	5.94573
French Franc	6.55957
Greek drachma	340.750
Irish Pound	0.787564
Italian Lire	1936.27
Luxembourg Franc	40.3399
Netherlands Guilder	2.20371
Portuguese Escudo	200.482
Spanish Peseta	166.386

例題

以下の例題はこのコマンドを使用して変換したものです:

```
$value:=10000 `フランスフラン値
  `ユーロ値に変換
$InEuros:=Euro converter($value;French Franc;Euro)
  `イタリアリラに変換
$InLires:=Euro converter($value;French Franc;Italian Lire)
```

Exp

Exp (number) -> 戻り値

引数	型		説明
number	実数	→	評価する数値
戻り値	実数	↩	指数関数値

説明

Expは自然対数の底(e = 2.71828...)のnumber乗の値を計算します。ExpはLogの逆の関数です。

Note: 4Dは定義済み定数e number (2.71828...) を提供しています。

例題

以下の例では、vrEに2.71828...が代入されます:

```
vrE:=Exp(1) ` vrE gets 2.17828...
```

Int

Int (number) -> 戻り値

引数	型	説明
number	実数 →	整数部を求める数値
戻り値	実数 ↩	整数部の数値

説明

Int は number の整数部を取り出して返します。number が負の場合は、ゼロから遠い数値に丸めます。

例題

以下の例は、負の数値と正の数値に対してIntがどのように機能するかを示しています。数値の少数点以下の部分が取り除かれている点に注目してください:

```
vlResult:=Int(123.4) ` vlResult gets 123
vlResult:=Int(-123.4) ` vlResult gets -124
```

Log

Log (number) -> 戻り値

引数	型		説明
number	実数	→	自然対数を求める数値
戻り値	実数	↩	自然対数値

説明

Log はnumberの自然対数を返します。LogはExpの逆の演算を実行する関数です。

Note: 4Dは定義済み定数e number (2.71828...) を提供しています。

例題

以下の行は1を表示します:

```
ALERT(String(Log(Exp(1))))
```

Mod

Mod (number1 ; number2) -> 戻り値

引数	型		説明
number1	倍長整数	→	除算される数値
number2	倍長整数	→	除算する数値
戻り値	実数	↩	余り

説明

Mod コマンドは、number1をnumber2で割り算し、その余りの整数を返します。

Notes:

- Modは整数、倍長整数、実数を受け入れます。しかしnumber1またはnumber2が実数の場合、それらの値は丸められてからMod計算を実行します。
- (2^31を超える) サイズの大きな実数を用いてModを使用する場合は注意が必要です。この場合、処理が標準的なプロセッサの計算能力の限界に達してしまう可能性があります。

なお余りを計算するためにモジューロ演算子 (%) を使用することもできます (を参照)。

警告: %演算子は整数または倍長整数式を用いて有効な結果を返します。実数値のモジューロを計算するには、Mod コマンドを使用しなければなりません。

例題

以下の例は、Modが異なる引数でどのように機能するかを紹介しています。各行はvlResultに値を代入し、コメント行にその結果を記述しています:

```
vlResult:=Mod(3;2) ` vlResult gets 1
vlResult:=Mod(4;2) ` vlResult gets 0
vlResult:=Mod(3.5;2) ` vlResult gets 0
```

Random

Random -> 戻り値

引数	型		説明
戻り値	倍長整数	←	乱数値

説明

Random は、0から32,767までの範囲の乱数を返します。

整数の範囲は、以下のように記述して変えることができます:

```
(Random%(vEnd-vStart+1))+vStart
```

vStartは範囲の開始値で、vEndは最終値です。

例題

以下の例は、10から30までの範囲の整数の乱数をvlResultに代入します:

```
vlResult:=(Random%21)+10
```

Round

Round (round ; places) -> 戻り値

引数	型		説明
round	実数	→	丸める数値
places	倍長整数	→	丸める小数部の位置
戻り値	実数	↩	placesで指定された場所で 丸められた数値

説明

Round は、指定されたplaces位置で数値を四捨五入します。

placesが正の数の場合、Roundの小数部を丸め、placesが負の場合には、整数部（小数点より左側）を丸めます。

placesで指定した桁位置に続く数字が5から9の場合、Roundが正のときは切り上げを、負の場合負の大きな値に丸めます。
placesで指定した桁位置に続く数字が0から4の場合、Roundは0に丸めます。

例題

下記は、さまざまな引数を使用してRoundの機能を示します。結果をvlResultに代入します。コメントは、変数vlResultに代入される値についての説明です:

```
vlResult:=Round(16.857;2) ` vlResult gets 16.86
vlResult:=Round(32345.67;-3) ` vlResult gets 32000
vlResult:=Round(29.8725;3) ` vlResult gets 29.873
vlResult:=Round(-1.5;0) ` vlResult gets -2
```

SET REAL COMPARISON LEVEL

SET REAL COMPARISON LEVEL (epsilon)

引数	型		説明
epsilon	実数	→	実数の同等性を比較するためのイプシロン値

説明

SET REAL COMPARISON LEVEL コマンドは、実数値と式の同等性を比較するために4Dが使用するイプシロン値を設定します。

コンピュータは常に実数を近似値で計算するため、実数の同等性をテストする時には、この近似値を考慮する必要があります。4Dは、実数を比較する時に2つの実数の差が一定の値より大きいかどうかをテストすることによって、近似値を確認します。この値は**イプシロン値**と呼ばれ、以下のように動作します:

2つの実数aとbがある時、Abs(a-b)がイプシロン値より大きい場合、これら2つの数値は等しくないとみなされます。それ以外の場合には等しいとみなされます。

デフォルトで、4Dはイプシロン値を10の-6乗（10^-6）に設定しています。イプシロン値は、常に正数を指定してください。例えば:

- 0.00001=0.00002 はFalseを返します。なぜなら違いは0.00001であり、これは10^-6より大きいからです。
- 0.000001=0.000002 はTrueを返します。なぜなら違いは0.000001であり、これは10^-6より大きくないからです。
- 0.000001=0.000003 rはFalseを返します。なぜなら違いは0.000002であり、これは10^-6より大きいからです。

SET REAL COMPARISON LEVELを使って、必要に応じて、エプシロン値を増大させるか、減少させることができます。

Note: 10^-6より小さい値が納められた数値タイプのインデックス付きフィールドに対して
クエリや並び替えを実行したい場合、インデックスの構築の前に必ずSET REAL
COMPARISON LEVELコマンドを実行してください。

警告: 通常、デフォルトのイプシロン値を変更するためにこのコマンドを使用する必要はありません。

重要: イプシロン値を変更しても、実数の同等性の比較に影響があるだけで、他の実数計算や実数値の表示には影響はありません。

Sin

Sin (number) -> 戻り値

引数	型		説明
number	実数	→	正弦を求めるラジアン値
戻り値	実数	⮐	正弦値

説明

Sin はnumberの正弦値を返します。numberはラジアンで指定します。

Note: 4DではPi、Degree、Radianという定数があらかじめ定義されています。Piはパイの数値 (3.14159...) を返し、Degreeはラジアンで表わされた1度 (0.01745...) を、Radianは度数で表わされた1ラジアン (57.29577...) を返します。

Square root

Square root (number) -> 戻り値

引数	型		説明
number	実数	→	平方根を求める数値
戻り値	実数	↩	平方根の値

説明

Square root は[number](number)の平方根を返します。

例題 1

以下の行は、値1.414213562373を$vrSquareRootOfTwoに代入します:

```
$vrSquareRootOfTwo :=Square root(2)
```

例題 2

以下のメソッドは三角形の斜辺長を返します。この三角形の2つの斜辺以外の辺は引数として渡されます:

```
  ` Hypotenuse method
  ` Hypotenuse ( real ; real ) -> real
  ` Hypotenuse ( legA ; legB ) -> Hypotenuse
C_REAL($0;$1;$2)
$0:=Square root(($1^2)+($2^2))
```

例えば、Hypotenuse (4;3) は5を返します。

Tan

Tan (number) -> 戻り値

引数	型		説明
number	実数	→	タンジェントを求めるラジアン値
戻り値	実数	↩	タンジェント値

説明

Tan はnumberの正接値を返します。numberはラジアンで指定します。

Note: 4DではPi、Degree、Radianという定数があらかじめ定義されています。Piはパイの数値 (3.14159...) を返し、Degreeはラジアンで表わされた1度 (0.01745...) を、Radianは度数で表わされた1ラジアン (57.29577...) を返します。

Trunc

Trunc (number ; places) -> 戻り値

引数	型	説明
number	実数	→ 切り捨てる数値
places	倍長整数	→ 切り捨てを行う位置
戻り値	実数	↩ 切り捨てられた数値

説明

Truncは、指定されたplacesの小数部を切り捨てた数値を返します。Truncは、常に元の値よりも小さい値を返します。

placesが正の数の場合は、numberの小数部を切り捨て、placesが負の場合には、整数部（小数点の左側）を切り捨てます。

例題

さまざまな引数を使用したTruncの機能を次に示します。結果をvlResultに代入します。コメントは、変数vlResultに代入される値についての説明です:

```
vlResult:=Trunc(216.897;1) ` vlResult gets 216.8
vlResult:=Trunc(216.897;-1) ` vlResult gets 210
vlResult:=Trunc(-216.897;1) ` vlResult gets -216.9
vlResult:=Trunc(-216.897;-1) ` vlResult gets -220
```

統計関数

- **統計関数**
- Average
- Max
- Min
- Std deviation
- Sum
- Sum squares
- Variance

統計関数

このテーマの関数は、一連の値に対する演算を行います。

Average, Max, Min, Sum, Sum squares, Std deviation そして Variance の各関数は、フィールドまたは配列を引数に取ります:

- フィールドの場合、カレントセレクションのレコードに適用されます。
- 配列の場合、配列要素に適用されます。

フィールドを使用する場合、Sum squares, Std deviation および Varianceは印刷時にのみ使用できます。

これらの関数はすべて数値データに対してのみ動作し、数値を返します。

統計関数をプリント処理以外で使用する

プリント処理以外でAverage, Max, Min および Sum関数をフィールドに対して使用すると、結果を算出するためにカレントセレクションのレコードをロードする必要があります。レコード数が多いと、処理に時間を必要とします。処理時間を減少させるためにはフィールドにインデックスを付けます。

注: 処理に時間がかかる場合、進捗サーモメータが表示されます。このサーモメータには停止ボタンがあり、ユーザは処理を中断できます。ユーザがこのボタンをクリックすると、OK変数に0が設定されます。処理が正しく完了すると、OK変数は1に設定されます。

統計関数をレポート印刷に使用する

統計関数がレポートで使用されると、レポート自身がそれぞれのレコードをロードするため、それらは特定の方法で実行されます。PRINT SELECTIONコマンド、あるいはデザインモードで**ファイル**メニューの**プリント...**を選択して印刷する場合に、フォームメソッドやオブジェクトメソッドの中でこれらの関数を使用します。

レポートに関数を使用する際に戻り値が信頼できるのは、ブレーク処理が有効の場合でかつブレークレベルが0の時だけです。つまりこれらの関数はレポートの最後、すべてのレコードを処理し終えた時にのみ有効です。

通常これらの関数は、B0ブレークエリアにある入力不可エリアのオブジェクトメソッド内でのみ使用します。

統計関数に引数として渡されるフィールドは、数値フィールドでなければならないことを覚えておいてください。

Average

Average (series) -> 戻り値

引数	型		説明
series	フィールド, 配列	→	平均を求めるデータ
戻り値	実数	↩	seriesの平均値

説明

Averageは、seriesの平均値を返します。seriesがインデックスフィールドの場合には、平均値を求めるためにインデックスが使用されます。

seriesに (1または2次元の) 配列を渡すこともできます。この場合配列は整数、倍長整数、または実数型でなければなりません。

このコマンドが正しく実行されると、OKシステム変数は1に設定されます。ユーザが進捗サーモメータの**停止**ボタンをクリックするなどして処理が中断されると、OK変数は0に設定されます。

例題 1

以下の例は、出力フォームのB0ブレークエリアにある変数vAverageに値を代入します。このコードは変数vAverageのオブジェクトメソッドです。オブジェクトメソッドは、レベル0のブレークが発生したときに実行されます:

```
vAverage:=Average([Employees] Salary)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます:

```
ALL RECORDS([Employees])
ORDER BY([Employees];[Employees]LastNm;>)
BREAK LEVEL(1)
ACCUMULATE([Employees]Salary)
FORM SET OUTPUT([Employees];"PrintForm")
PRINT SELECTION([Employees])
```

Note: BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は印刷コマンドを参照してください。

例題 2

この例題ではセレクション中トップ15の平均点を計算します:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!11/07/01!)
ORDER BY([Exams];[Exams]Exam_Grade;<)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
ARRAY REAL($ArrGrades;15)
vAverage:=Average($ArrGrades)
```

Max

Max (series) -> 戻り値

引数	型		説明
series	フィールド, 配列	→	最大値を求めるデータ
戻り値	実数	↩	series中の最大値

説明

Maxは、series中の最大値を返します。seriesがインデックスフィールドの場合には、最大値を求めるためにインデックスが使用されます。

seriesに (1または2次元の) 配列を渡すこともできます。この場合配列は整数、倍長整数、または実数型でなければなりません。

seriesセレクションが空の場合、**Max**は0を返します。

このコマンドが正しく実行されると、OKシステム変数は1に設定されます。ユーザが進捗サーモメータの**停止**ボタンをクリックするなどして処理が中断されると、OK変数は0に設定されます。

例題 1

以下の例は、出力フォームのB0ブレークエリアにある変数vMaxに値を代入します。変数はレポートの最後に印刷されます。オブジェクトメソッドは変数にフィールドの最大値を代入し、レポートの最後のブレークで印刷されます。

```
vMax:=Max([Employees] Salary)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます:

```
ALL RECORDS([Employees])
ORDER BY([Employees];[Employees]LastNm;>)
BREAK LEVEL(1)
ACCUMULATE([Employees]Salary)
FORM SET OUTPUT([Employees];"PrintForm")
PRINT SELECTION([Employees])
```

注: BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は印刷コマンドを参照してください。

例題 2

この例題では配列中の最も大きな数を取得します:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!11/07/01!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vMax:=Max($ArrGrades)
```

Min

Min (series) -> 戻り値

引数	型		説明
series	フィールド, 配列	→	最小値を求めるデータ
戻り値	実数	←	series中の最小値

説明

Minは、series中の最小値を返します。seriesがインデックスフィールドの場合には、最小値を求めるためにインデックスが使用されます。

seriesセレクションが空の場合、**Min**は0を返します。

seriesに (1または2次元の) 配列を渡すこともできます。この場合配列は整数、倍長整数、または実数型でなければなりません。

このコマンドが正しく実行されると、OKシステム変数は1に設定されます。ユーザが進捗サーモメータの**停止**ボタンをクリックするなどして処理が中断されると、OK変数は0に設定されます。

例題 1

以下の例は、出力フォームのB0ブレークエリアにある変数vMinに値を代入します。変数はレポートの最後に印刷されます。オブジェクトメソッドは変数にフィールドの最小値を代入し、レポートの最後のブレークで印刷されます:

```
vMin:=Min([Employees]Salary)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます:

```
ALL RECORDS([Employees])
ORDER BY([Employees];[Employees]LastNm;>)
BREAK LEVEL(1)
ACCUMULATE([Employees]Salary)
FORM SET OUTPUT([Employees];"PrintForm")
PRINT SELECTION([Employees])
```

Note: BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は印刷コマンドを参照してください。

例題 2

以下の例題は従業員の最低売上高を検索し、警告ダイアログに結果を表示します:

```
ALERT("Minimum sale = "+String(Min([Employees]Sales)))
```

例題 3

この例題では配列中で最も小さい値を取得します:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!11/07/01!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vMin:=Min($ArrGrades)
```

Std deviation

Std deviation (series) -> 戻り値

引数	型		説明
series	フィールド, 配列	→	標準偏差を求めるデータ
戻り値	実数	↩	seriesの標準偏差

説明

Std deviationは、seriesの標準偏差を返します。seriesがインデックスフィールドの場合に、標準偏差を求めるためにインデックスが使用されます。

seriesに (1または2次元の) 配列を渡すこともできます。この場合配列は整数、倍長整数、または実数型でなければなりません。

例題 1

以下の例は変数vDeviateのオブジェクトメソッドです。オブジェクトメソッドはvDeviateに一連のデータの標準偏差を代入します:

```
vDeviate:=Std deviation([Table1]DataSeries)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます:

```
ALL RECORDS([Table1])
ORDER BY([Table1];[Table1]DataSeries;>)
BREAK LEVEL(1)
ACCUMULATE([Table1]DataSeries)
OUTPUT FORM([Table1];"PrintForm")
PRINT SELECTION([Table1])
```

注: BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は印刷コマンドを参照してください。

例題 2

この例題では配列に格納された一連の値の標準偏差を求めます:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!11/07/01!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vStdDev:=Std deviation($ArrGrades)
```

Sum

Sum (series) -> 戻り値

引数	型		説明
series	フィールド, 配列	→	合計を求めるデータ
戻り値	実数	↩	seriesの合計

説明

Sumコマンドは、seriesの合計値を返します。

seriesがインデックスフィールドの場合、合計を求めるためにインデックスが使用されます。

(一次元または二次元の) 配列を series に渡すことができます。この場合配列は整数、倍長整数、または実数型でなければなりません。

このコマンドが正しく実行されると、OKシステム変数は1に設定されます。ユーザが進捗サーモメータの**停止**ボタンをクリックするなどして処理が中断されると、OK変数は0に設定されます。

例題 1

以下の例はフォームに置かれた変数vTotalのオブジェクトメソッドです。オブジェクトメソッドはvTotalに一連のデータの合計値を代入します:

```
vTotal:=Sum([Employees]Salary)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます:

```
ALL RECORDS([Employees])
ORDER BY([Employees];[Employees]LastNm;>)
BREAK LEVEL(1)
ACCUMULATE([Employees]Salary)
OUTPUT FORM([Employees];"PrintForm")
PRINT SELECTION([Employees])
```

注: BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は印刷コマンドを参照してください。

例題 2

この例では配列中の値の合計値を求めます:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vSum:=Sum($ArrGrades)
```

Sum squares

Sum squares (series) -> 戻り値

引数	型	説明
series	フィールド, 配列	→ 平方和を求めるデータ
戻り値	実数	↩ seriesの平方和

説明

Sum squaresは、seriesの平方和を返します。

seriesがインデックスフィールドの場合、標準偏差を求めるためにインデックスが使用されます。

(一次元または二次元の) 配列を series に渡すことができます。この場合配列は整数、倍長整数、または実数型でなければなりません。

例題 1

以下の例は変数vSquaresのオブジェクトメソッドです。オブジェクトメソッドはvSquaresに一連のデータの平方和を代入します。vSquares変数はレポートの最後のブレークで印刷されます:

```
vSquares:=Sum squares([Table1]DataSeries)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます:

```
ALL RECORDS([Table1])
ORDER BY([Table1];[Table1]DataSeries;>)
BREAK LEVEL(1)
ACCUMULATE([Table1]DataSeries)
OUTPUT FORM([Table1];"PrintForm")
PRINT SELECTION([Table1])
```

注: BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は印刷コマンドを参照してください。

例題 2

この例題では配列に格納された一連の値の平方和を取得します:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vSumSquares:=Sum squares($ArrGrades)
```

Variance

Variance (series) -> 戻り値

引数	型		説明
series	フィールド, 配列	→	分散を求めるデータ
戻り値	実数	⮐	seriesの分散

説明

Varianceは、seriesの分散を返します。

seriesがインデックスフィールドの場合、分散を求めるためにインデックスが使用されます。

(一次元または二次元の) 配列を series に渡すことができます。この場合配列は整数、倍長整数、または実数型でなければなりません。

分散は一連の値が期待値からどれだけ散らばっているかを示す値で、平均からの分散を計測します。4Dは以下の分散計算式を使用します:

Variance(x) = Sum (x-m)*(x-m)/(n-1)
m = 平均
n = 標本数

考慮される値が標本でない場合、Varianceから返される値を (n-1)/n でかけ合わせます。

例題 1

以下の例は変数varのオブジェクトメソッドです。オブジェクトメソッドはvarに一連のデータの分散を代入します:

```
var:=Variance(Students]Grades)
```

以下のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます:

```
ALL RECORDS([Students])
ORDER BY([Students];[Students]Class;>)
BREAK LEVEL(1)
ACCUMULATE([Students]Grades)
OUTPUT FORM([Students];"PrintForm")
PRINT SELECTION([Students])
```

注: BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は印刷コマンドを参照してください。

例題 2

この例題では配列に格納された値の分散を求めます:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vVariance:=Variance($ArrGrades)
```

通信

- GET SERIAL PORT MAPPING
- RECEIVE BUFFER
- RECEIVE PACKET
- RECEIVE RECORD
- RECEIVE VARIABLE
- SEND PACKET
- SEND RECORD
- SEND VARIABLE
- SET CHANNEL
- SET TIMEOUT
- USE CHARACTER SET

GET SERIAL PORT MAPPING

GET SERIAL PORT MAPPING (numArray ; nameArray)

引数	型	説明
numArray	倍長整数配列	← ポート番号の配列
nameArray	文字配列	← ポート名の配列

説明

GET SERIAL PORT MAPPING コマンドはマシンのシリアルポート番号とシリアルポート名を、2つの配列numArray と nameArrayに返します。

Mac OS Xでは、USBシリアルアダプターの使用時に、OSがポート番号を動的に割り当てるため、このコマンドが役立ちます。シリアルポートの実際のIDとは関係なく、その名前（固定）を用いてシリアルポートを取り扱うことができます。

Note: このコマンドは標準ポートでは、意味ある値を返しません。標準ポートを使用する場合は、SET CHANNELコマンドに旧方式である0または1のポート番号を指定してください。

例題

このプロジェクトメソッドは、割り当てられたシリアルポート番号に関係なく同じシリアルポートを取得するために使用できます:

```
ARRAY TEXT($arrPortNames;0)
ARRAY LONGINT($arrPortNums;0)
C_LONGINT($vPortNum)

 `シリアルポートの現在の番号を取得
GET SERIAL PORT MAPPING($arrPortNums;$arrPortNames)
$vPortNum:=Find in array($arrPortNames;vPortName)
 ` vPortName には使用するポートの名前が格納されている;
 ` この値はダイアログで設定したり、フィールドに格納されている
If($vPortNum>0)
   SET CHANNEL($vPortNum+100;params) `params には通信パラメタが格納されている
 `拡張ポートを示す100を加えることを忘れないでください。
End if
... `処理を実行
SET CHANNEL(11) `ポートを閉じる
```

RECEIVE BUFFER

RECEIVE BUFFER (receiveVar)

引数	型		説明
receiveVar	変数	→	データを受信する変数

説明

RECEIVE BUFFER は、SET CHANNELコマンドで前もって開いたシリアルポートからデータを読み込みます。シリアルポートは、コマンドで読み込まれるまで内容を保持するバッファを持ちます。**RECEIVE BUFFER**はシリアルバッファから文字を読み込み、receiveVarに格納して、バッファを消去します。バッファ中に文字が存在しなければ、receiveVarは何も含みません。

Windows

Windowsのシリアルポートバッファのサイズは10KBに制限されています。つまり、バッファがオーバーフローする可能性があるということです。バッファがいっぱいになった後、新しい文字を受信すると、最も古いものと置き換えられます。古くなった文字は失われるため、新しい文字を受信する際は、すみやかにバッファを読み込むことが重要です。

Mac OS

Mac OS Xのシリアルポート バッファーサイズは、理論的には制限がありません。実際には利用可能なメモリ量に制限されます。バッファーがいっぱいになった後、新しい文字を受信すると、最も古いものと置き換えられます。古くなった文字は失われるため、新しい文字を受信する際は、すみやかにバッファを読み込むことが重要です。

RECEIVE BUFFERコマンドはRECEIVE PACKETと異なり、バッファ中のデータが何であれ、それを即座に返します。RECEIVE PACKETはバッファ中に指定した文字を発見するまで、または指定した数の文字が入力されるまで待ちます。

RECEIVE BUFFERの実行中に、Ctrl-Alt-Shift (Windows) または Command-Option-Shift (Macintosh) を押して、受信を中断できます。中断することにより、エラー-9994が生成されます。ON ERR CALLを使用してインストールしたエラー処理メソッドにより、このエラーをとらえることができます。

例題

以下のプロジェクトメソッドLISTEN TO SERIAL PORTは、RECEIVE BUFFERコマンドを使用してシリアルポートからテキストを取得し、それをインタープロセス変数に追加します:

```
  ` LISTEN TO SERIAL PORT
  ` シリアルポートを開く
SET CHANNEL(201;Speed 9600+Data bits 8+Stop bits One+Parity None)
<>IP_Listen_Serial_Port:=True
While(<>IP_Listen_Serial_Port)
   RECEIVE BUFFER($vtBuffer)
   If((Length($vtBuffer)+Length(<>vtBuffer))>MAXTEXTLEN)
      <>vtBuffer:=""
   End if
   <>vtBuffer:=<>vtBuffer+$Buffer
End while
```

この時点で、他のプロセスからインタープロセス変数<>vtBufferを読み込み、シリアルポートから受信したデータの処理を行うことができます。

シリアルポートの監視を中断するには、以下のコードを実行します:

```
  ` シリアルポートの待ち受けを停止する
<>IP_Listen_Serial_Port:=False
```

プロセス間でのコンフリクトを避けるために、セマフォを利用してインタープロセス変数<>vtBufferへのアクセスを保護する必要がある点に注意してください。詳細はSemaphoreコマンドを参照してください。

RECEIVE PACKET

RECEIVE PACKET ({DocRef ;} receiveVar ; stopChar | numBytes)

引数	型		説明
DocRef	DocRef	→	ドキュメント参照番号, または カレントチャンネル (シリアルポートまたはドキュメント)
receiveVar	テキスト変数, BLOB変数	→	データを受け取る変数
stopChar | numBytes	文字, 倍長整数	→	受信を停止する文字, または受信するバイト数

説明

RECEIVE PACKET コマンドは、シリアルポートまたはドキュメントから文字を読み込みます。

docRefを指定した場合、このコマンドは Open document, Create document または Append document で開かれたドキュメントからデータを取得します。docRefを指定しない場合、このコマンドは SET CHANNELコマンドで開かれたシリアルポートかドキュメントからデータを取得します。

読み込み元に関わらず、読み込まれた文字は、テキスト、文字、またはBLOB型のreceiveVar変数に返されます。文字が SEND PACKET コマンドで送信された場合、型はパレットが送信された際の型に対応しなければなりません。

Notes:

- 受信したパケットがBLOB型の場合、コマンドは USE CHARACTER SET コマンドで指定された文字セットを考慮しません。BLOBには変更されないデータが返されます。
- テキスト型のパケットを受信した場合、RECEIVE PACKET コ マンドはByte Order Marks (BOM) をサポートします。この場合、カレントの文字セットがUnicodeタイプ (UTF-8, UTF-16またはUTF-32) であれば、4Dは受信した先頭のバイトでBOMの識別を試みます。BOMが検知されると、それはreceiveVar変数から取り除かれ、4Dは現在の文字セットではなくBOMで指定された文字セットを使用します。

特定の文字数まで読み込むためには、numBytesにその数を渡します。receiveVarがテキスト型の場合、一回の呼び出しで、Unicodeモードでは2GB (理論的な値) までのテキストを読み込めます。

特定の文字列（1桁以上の文字で構成される）が現われるまでデータを取り込むには、stopCharにその文字列を渡します（この文字列はreceiveVarに含まれません）。

この場合、stopCharで指定した文字を見つけることが出来なければ:

- RECEIVE PACKET がドキュメントを読み込むとき、ドキュメントの最後で読み込みを終了します。
- RECEIVE PACKET がシリアルポートから読み込むとき、タイムアウト (SET TIMEOUT参照) に達するか、または利用者が割込（以下を参照）をかけるまで待ち続けます。

RECEIVE PACKET の実行中、利用者はCtrl＋Alt＋Shift (Windows) またはコマンド＋Option＋Shift (Macintosh)キーを押下することで、割込をかけることが出来ます。割込が発生するとエラー-9994が生成され、ON ERR CALLでインストールされたエラー処理メソッドでとらえることができます。通常、シリアルポートで通信している場合のみ、割込を処理します。

ドキュメントを読み込む場合、最初のRECEIVE PACKET コマンドは、ドキュメントファイルの先頭から読み込みを開始します。その後のデータ読み込みは、最後に読み込まれたバイトの次から開始します。

Note: このコマンドは、SET CHANNEL を用いて開かれたドキュメントに対して有効です。一方で、Open document, Create document または Append documentで開かれたドキュメントに関しては、Get document position または SET DOCUMENT POSITION コマンドを使用して、次の書き込み (SEND PACKET) や読み込み(RECEIVE PACKET) を行うドキュメント中の位置を取得したり設定したりできます。

ファイルの最後を越えて読み込もうとした場合、RECEIVE PACKET は、そのポイントまでに読み込んだデータを返し、システム変数OKに1を代入します。その次の RECEIVE PACKET は空の文字列を返し、システム変数OKに0を代入します。

例題 1

以下の例は、20バイトのデータをシリアルポートから読み込み、変数getTwentyに格納します:

```
RECEIVE PACKET(getTwenty;20)
```

例題 2

以下の例は、変数myDocで参照されるドキュメントからデータを読み込み、変数vDataに格納します。ここでは改行が見つかるまで読み込みます:

```
RECEIVE PACKET(myDoc;vData;Char(Carriage return))
```

例題 3

以下の例は、変数myDocで参照されるドキュメントからデータを読み込み、変数vDataに格納します。HTMLタグ</TD>(テーブルセルの終わり)が現われるまでデータを読み込みます:

```
RECEIVE PACKET(myDoc;vData;"</TD>")
```

例題 4

以下の例は、ドキュメントファイルから読み込んだデータをフィールドに格納します。データは、固定長で格納されています。このメソッドは、サブルーチンを呼び出してデータの後ろに付随する不要なスペースを取り除きます:

```
$vhDocRef :=Open document("";"TEXT") ` TEXTドキュメントを開く
If(OK=1) ` ドキュメントが開かれたら
   REPEAT ` データがなくなるまで繰り返す
      RECEIVE PACKET($vhDocRef;$Var1;15) ` 15文字読み込む
      RECEIVE PACKET($vhDocRef;$Var2;15) ` 2番目のフィールドに同じことを行う
      If(($Var1#"")|($Var2#"")) ` どちらかのフィールドが空でなければ
         CREATE RECORD([People]) ` レコードを作成
         [People]First :=Strip($Var1) ` 名を格納
         [People]Last :=Strip($Var2) ` 姓を格納
         SAVE RECORD([People]) ` レコードを保存
      End if
   Until(OK=0)
   CLOSE DOCUMENT($vhDocRef) ` ドキュメントを閉じる
End if
```

データの終わりのスペースは以下のStripメソッドで取り除きます:

```
For($i;Length($1);1;-1) ` 文字の最後からループ
   If($1[[$i]]#" ") ` スペースでなければ...
      $i :=-$i  ` ループを終了する
   End if
End for
$0:=Delete string($1;-$i;Length($1)) ` スペースを削除
```

システム変数およびセット

RECEIVE PACKET呼び出し後、エラーなしでパケットを受信すればOKシステム変数に1が、そうでなければ0が設定されます。

RECEIVE RECORD

RECEIVE RECORD {(aTable)}
引数　型　　説明
aTable テーブル→レコードを受信するテーブル, または 省略した場合デフォルトテーブル

説明

RECEIVE RECORDは、SET CHANNELコマンドで開いたシリアルポートまたはドキュメントからtableにレコードを受信します。受信するレコードはSEND RECORDで送信したものでなければなりません。RECEIVE RECORDを実行すると、tableに新しいレコードが自動で作成されます。レコードを正しく受信した時点で、SAVE RECORDコマンドを使用して新しいレコードを保存します。

完全なレコードを受信します。つまりレコードにあるいはレコードとともに格納されたピクチャやBLOBも受信します。

重要: SEND RECORD と RECEIVE RECORDを使用してレコードが送受信される場合、送信元と送信先のテーブル構造は互換性のあるものでなくてはなりません。互換性がない場合、RECEIVE RECORDコマンドの実行時4Dがテーブル定義に応じて値を変換します。

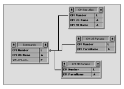
Notes:

1. このコマンドを使用してドキュメントからレコードを受信する場合、ドキュメントはSET CHANNELコマンドを使用して開かれていなければなりません。Open document, Append document または Create documentで開かれたドキュメントに対して、RECEIVE RECORDを使用することはできません。
2. RECEIVE RECORDの実行中、Ctrl-Alt-Shift (Windows) またはCommand-Option-Shift (Macintosh) を押して、受信を中断することができます。中断することにより、エラー-9994が生成されます。ON ERR CALLを使用してインストールされたエラー処理メソッドにより、このエラーをとらえることができます。通常、シリアルポート経由での通信の場合にのみ、受信の中断処理を実行する必要があります。

例題

データのアーカイブ作成や、異なる場所で使用されている同じシングルユーザデータベース間でデータをやり取りする際には、SEND VARIABLE, SEND RECORD, RECEIVE VARIABLE そして RECEIVE RECORDコマンドを組み合わせて使用するとよいでしょう。EXPORT TEXT や IMPORT TEXT等の読み込み/書き出しコマンドを利用して、4Dデータベース間でデータ交換を実行できます。しかしデータ中にグラフィックやリレートテーブルが含まれる場合には、SEND RECORD と RECEIVE RECORDを使う方がはるかに便利です。

例えば、4Dと4D Writeを使用して作成されたドキュメントシステムを考えてみます。別々の場所にいる複数の製作者が作業を実行するため、異なるデータベース間でデータをやり取りする簡単な方法が必 要となりました。以下の図はこのデータベースストラクチャを簡単に表わしたものです:



テーブル[Commands]には、各コマンドやトピックに関する説明が納められます。テーブル[CM US Params]および[CM FR Params]にはそれぞれ、英語版またはフランス語版の各コマンドに対する引数のリストが納められます。また、テーブル[CM See Also]には、各コマンドに対する参照としてリストされるコマンドが納められています。したがって、データベース間でドキュメントのやり取りを実行するには、[Commands]レコードとそれにリレートするレコードを送信しなければなりません。これを実行するには、SEND RECORDとRECEIVE RECORDコマンドを使用しています。さらにSEND VARIABLEとRECEIVE VARIABLEコマンドを使い、読み込み／書き出しドキュメントにタグを付けています。

ドキュメントの書き出しを実行する簡略化したプロジェクトメソッドを以下に示します:

```
  // CM_EXPORT_SEL プロジェクトメソッド
  // このメソッドは [Commands] テーブルのカレントセレクションを処理対象とする

 SET CHANNEL(12;"") ` ユーザにドキュメントを作成させ、チャンネルを開く
 If(OK=1)
  // 内容を示すタグを変数でドキュメントに指定する
  // Note: BUILD_LANG プロセス変数は送信するデータの言語、US (English) または FR (French)を示す
    $vsTag:="4DV6COMMAND"+BUILD_LANG
    SEND VARIABLE($vsTag)
  // 書き出す [Commands] レコード数を示す変数を送信
    $vlNbCmd:=Records in selection([Commands])
    SEND VARIABLE($vlNbCmd)
    FIRST RECORD([Commands])
  // コマンドごとに
    For($vlCmd;1;$vlNbCmd)
  // [Commands] レコードを送信
       SEND RECORD([Commands])
  // リレートするレコードを選択
       RELATE MANY([Commands])
  // ランゲージごとに、引数の数をしめす変数を送信
       Case of
          :(BUILD_LANG="US")
             $vlNbParm:=Records in selection([CM US Params])
          :(BUILD_LANG="FR")
             $vlNbParm:=Records in selection([CM FR Params])
       End case
       SEND VARIABLE($vlNbParm)
  // parameter レコードがあれば送信
       For($vlParm;1;$vlNbParm)
          Case of
             :(BUILD_LANG="US")
                SEND RECORD([CM US Params])
                NEXT RECORD([CM US Params])
             :(BUILD_LANG="FR")
                SEND RECORD([CM FR Params])
                NEXT RECORD([CM FR Params])
          End case
       End for
  // "See Also"の数を示す変数を送信
       $vlNbSee:=Records in selection([CM See Also])
       SEND VARIABLE($vlNbSee)
  // [See Also] レコードがあれば送信
       For($vlSee;1;$vlNbSee)
          SEND RECORD([CM See Also])
          NEXT RECORD([CM See Also])
       End for
  // 次の[Commands] レコード
       NEXT RECORD([Commands])
    End for
    SET CHANNEL(11) ` Close the document
 End if
```

ドキュメントの読み込みを実行する簡略化したプロジェクトメソッドを以下に示します:

```
  ` CM_IMPORT_SEL プロジェクトメソッド

 SET CHANNEL(10;"") // ユーザに読み込むドキュメントを選択させる
 If(OK=1) ` ドキュメントが開かれれば
    RECEIVE VARIABLE($vsTag) // タグ変数を読み込む
    If($vsTag="4DV6COMMAND@") // 正しいタグを読み込んだかチェック
       $CurLang:=Substring($vsTag;Length($vsTag)-1) // タグからランゲージを取り出す
       If(($CurLang="US")&NBSP;|&NBSP;($CurLang="FR")) // 有効なランゲージなら
          RECEIVE VARIABLE($vlNbCmd) // いくつのコマンドが格納されているか
          If($vlNbCmd>0) ` 1つ以上存在すれば
             For($vlCmd;1;$vlNbCmd) // アーカイブされた [Commands] レコード毎に
  // レコードを受信
                RECEIVE RECORD([Commands])
  // 新規レコードまたは既存のレコードへの上書きを保存するサブルーチンの呼び出し
                CM_IMP_CMD($CurLang)
  // パラメタ数の読み込み
                RECEIVE VARIABLE($vlNbParm)
                If($vlNbParm>=0)
  // RECEIVE RECORDを使用して新規レコードまたは既存の
  // レコードへの上書きを保存するサブルーチンの呼び出し
                   CM_IMP_PARM($vlNbParm;$CurLang)
                End if
  // "See Also"の数を読み込み
                RECEIVE VARIABLE($vlNbSee)
                If($vlNbSee>0)
  // RECEIVE RECORDを使用して新規レコードまたは
  // 既存のレコードへの上書きを保存するサブルーチンの呼び出し
                   CM_IMP_SEEA($vlNbSee;$CurLang)
                End if
             End for
          Else
             ALERT("この書き出しドキュメント内のコマンド数が無効です。")
          End if
       Else
          ALERT("この書き出しドキュメントの言語が正しくありません。")
       End if
    Else
       ALERT("これはコマンド書き出しドキュメントではありません。")
    End if
    SET CHANNEL(11) // ドキュメントを閉じる
 End if
```

データ受信中にシステム変数OKの評価も行わず、またエラーの検出も行っていない点に注意してください。しかし、ドキュメントそのものを表わす変数をドキュメントに保存するため、これらの変数がいったん受信され意味を持つ場合には、エラーの可能性は低くなります。例えば、ユーザが誤ったドキュメントをオープン した場合、この処理は最初の判定式で即座に中断されます。

システム変数およびセット

レコードを受信するとシステム変数OKは1に、そうでなければ0に設定されます。

RECEIVE VARIABLE

RECEIVE VARIABLE (variable)

引数	型	説明
variable	変数	➡ データを受信する変数

説明

RECEIVE VARIABLE は、SET CHANNELで開いたシリアルポートまたはドキュメントから、SEND VARIABLEコマンドで送信したvariableを受信します。

インタプリタモードでは、このコマンドのコール前に変数が存在しない場合、変数が作成され、受信内容に応じたタイプ付けが行われます。コンパイル後モードでは、変数のタイプは受信するものと同じでなくてはなりません。両方の場合とも、変数の内容は受信した値で置き換えられます。

インタプリタモードでは、RECEIVE VARIABLEの呼び出し前に変数が存在しない場合、変数が作成され、受信内容に応じた型付けが行われ、代入されます。コンパイルモードでは、変数のタイプは受信するものと同じでなくてはなりません。両方の場合とも、変数の内容は受信した値で置き換えられます。

Notes:

1. このコマンドを使用してドキュメントから変数を受信する場合、ドキュメントはSET CHANNELコマンドを使用して開かれていなければなりません。Open document, Append document または Create documentで開かれたドキュメントに対して、RECEIVE VARIABLEを使用することはできません。
2. このコマンドは配列をサポートしません。ドキュメントやシリアルポートを介して配列を送受信したい場合、BLOBコマンドを使用してください。
3. RECEIVE VARIABLEの実行中、Ctrl-Alt-Shift (Windows) またはCommand-Option-Shift (Macintosh) を押して、受信を中断することができます。中断することにより、エラー-9994が生成されます。ON ERR CALLを使用してインストールされたエラー処理メソッドにより、このエラーをとらえることができます。通常、シリアルポート経由での通信の場合にのみ、受信の中断処理を実行する必要があります。

例題

RECEIVE RECORD コマンドの例題を参照

システム変数およびセット

変数を受信するとシステム変数OKに1が設定され、そうでなければ0が設定されます。

SEND PACKET

SEND PACKET ({DocRef ;} packet)

引数	型	説明
DocRef	DocRef	→ ドキュメント参照番号, または カレントのチャンネル (シリアルポートまたはドキュメント)
packet	文字, BLOB	→ 送信する文字またはBLOB

説明

SEND PACKET は、シリアルポートやドキュメントにパケットを送ります。docRefを指定した場合、パケットはdocRefで参照されるドキュメントに書き込まれます。docRefを指定しない場合、あらかじめ SET CHANNEL コマンドで開かれたシリアルポートまたはドキュメントにパケットを書き込みます。

packetはデータの一部であり、一般的には文字列です。
packetにBLOBを渡すこともできます。これにより、テキストモードで送信される文字のエンコーディングに関連する制約を回避できます (例題 2参照)。

Note: packetにBLOBを渡す場合、コマンドは USE CHARACTER SET コマンドで定義された文字セットを考慮しません。BLOBは変更されずに送信されます。

SEND PACKETコマンドを使用する前に、SET CHANNEL コマンドでシリアルポートやドキュメントを開くか、ドキュメントコマンドを使用してドキュメントを開かなければなりません。

既存のドキュメントに書き込む場合は、ドキュメントが USE CHARACTER SET で開かれていない限り、最初のSEND PACKET コマンドはドキュメントの先頭から書き始めます。それに続くSEND PACKET コマンドは、ドキュメントファイルが閉じられるまでパケットを後ろに書き加えます。

Note: このコマンドは SET CHANNEL で開かれたドキュメントに対して有効です。一方で、Open document, Create document あるいは Append document で開かれたドキュメントに関しては、Get document position や SET DOCUMENT POSITION コマンドを使用して、次の書き込み (SEND PACKET) や読み込み (RECEIVE PACKET) を行うドキュメント中の位置を取得したり設定したりできます。

例題 1

以下の例は、フィールドのデータをドキュメントに書き込みます。この例では、固定長データとして書き込みます。固定長フィールドは常に決まった長さです。フィールドが指定した長さよりも短い場合、その分のスペースを埋め込みます（つまり、指定された長さになるまでスペースを付け加えます）。固定長データの使用は、データ格納の合理的な方法とはいえませんが、一部のコンピュータシステムやアプリケーションでは、まだ使用されています:

```
$vhDocRef :=Create document("") ` ドキュメントを作成
If(OK=1) ` ドキュメントが作成されたら
   For($vlRecord;1;Records in selection([People])) ` レコードごとに繰り返す
  ` パケット送信。名フィールドのデータを含む長さ15のパケットを作成
      SEND PACKET($vhDocRef;Change string(15*Char(SPACE);[People]First;1))
  ` 2つ目のパケット送信。姓フィールドのデータを含む長さ15のパケットを作成
      SEND PACKET($vhDocRef;Change string(15*Char(SPACE);[People]Last;1))
      NEXT RECORD([People])
   End for
  ` Char(26)を送信、これはいくつかのコンピュータでend-of-fileのマーカとして使用されます
   SEND PACKET($vhDocRef;Char(SUB ASCII code))
   CLOSE DOCUMENT($vhDocRef) ` ドキュメントを閉じる
End if
```

例題 2

この例題はBLOB経由でドキュメントの拡張文字を送信したり受信したりする方法を示します:

```
C_BLOB($send_blob)
C_BLOB($receive_blob)
TEXT TO BLOB("azerty";$send_blob;UTF8 Text without length)
SET BLOB SIZE($send_blob;16;255)
$send_blob{6}:=0
$send_blob{7}:=1
$send_blob{8}:=2
$send_blob{9}:=3
$send_blob{10}:=0
$vlDocRef:=Create document("blob.test")
If(OK=1)
   SEND PACKET($vlDocRef;$send_blob)
   CLOSE DOCUMENT($vlDocRef)
End if
$vlDocRef:=Open document(document)
If(OK=1)
   RECEIVE PACKET($vlDocRef;$receive_blob;65536)
   CLOSE DOCUMENT($vlDocRef)
End if
```

SEND RECORD

SEND RECORD {(aTable)}

引数	型	説明
aTable	テーブル	→ カレントレコードを送信するテーブル, または 省略した場合デフォルトテーブル

説明

SEND RECORD は、aTableのカレントレコードをSET CHANNELコマンドで開いたシリアルポートまたはドキュメントに送信します。レコードは特別な内部フォーマットで送信され、RECEIVE RECORDでのみ読み込むことができます。カレントレコードが存在しなければ、SEND RECORDは何も行いません。

完全なレコードを送信します。つまりレコードにあるいはレコードとともに格納されたピクチャやBLOBも送信します。

重要: SEND RECORD と RECEIVE RECORDを使用してレコードが送受信される場合、送信元と送信先のテーブル構造は互換性のあるものでなくてはなりません。互換性がない場合、RECEIVE RECORDコマンドの実行時4Dがテーブル定義に応じて値を変換します。

Notes: このコマンドを使用してドキュメントにレコードを送信する場合、ドキュメントはSET CHANNELコマンドを使用して開かれていなければなりません。Open document, Append document または Create documentで開かれたドキュメントに対して、SEND RECORDを使用することはできません。

互換性に関する注意: 4Dバージョン11より、このコマンドはサブテーブルをサポートしません。

例題

RECEIVE RECORDコマンドの例題を参照

SEND VARIABLE

SEND VARIABLE (variable)

引数	型	説明
variable	変数	⇒ 送信する変数

説明

SEND VARIABLE は、SET CHANNELで開いたシリアルポートまたはドキュメントにvariableを送信します。変数はRECEIVE VARIABLEでなければ読み込むことのできない特別な内部フォーマットで送信されます。SEND VARIABLEは、(型と値を含む)完全な変数を送信します。

Notes:

1. このコマンドを使用してドキュメントに変数を送信する場合、ドキュメントはSET CHANNELコマンドを使用して開かれていなければなりません。Open document, Append document または Create documentで開かれたドキュメントに対して、SEND VARIABLEを使用することはできません。
2. このコマンドは配列変数をサポートしません。ドキュメントやシリアルポート経由で配列の送受信を行いたい場合は、を使用してください。.

例題

RECEIVE RECORDの例題参照

SET CHANNEL

SET CHANNEL (port ; settings)

引数	型		説明
port	倍長整数	⇨	シリアルポート番号
settings	倍長整数	⇨	シリアルポート設定

SET CHANNEL (operation ; document)

引数	型		説明
operation	倍長整数	⇨	行うドキュメント処理
document	文字	⇨	ドキュメント名

説明

SET CHANNEL コマンドはシリアルポートまたはドキュメントを開きます。このコマンドでは、同時に1つのポートまたは1つのドキュメントファイルしか開くことができません。開いたシリアルポートを閉じるにはSET CHANNEL(11)を用います。

Historical Note: このコマンドはもともと、シリアルポートやディスク上のドキュメントを用いて作業を実行するために使用された最初の4Dコマンドです。以来、新しいコマンドが追加されています。現在では、ディスク上のドキュメントを使った作業を実行する際には、Open document, Create document そして Append documentといったコマンドを使用します。これらのコマンドを利用し、SEND PACKET や RECEIVE PACKETを使用してドキュメントに文字を書き込んだり、読み込むことができます（この2つのコマンドもSET CHANNELとともに使用できます）。しかし、SEND VARIABLE, RECEIVE VARIABLE, SEND RECORD そして RECEIVE RECORDコマンドを使用したい場合には、ディスクのドキュメントにアクセスする際にSET CHANNELを使用しなくてはなりません。

SET CHANNELの説明は、2つの部分からなります:

- シリアルポートに使用
- ドキュメントに使用

シリアルポートに使用 - SET CHANNEL (port;settings)

SET CHANNEL コマンドの第一の形式はシリアルポートを開き、プロトコルや他のポート情報を設定します。データの送信は、SEND PACKET, SEND RECORD または SEND VARIABLEで行えます。データの受信は、RECEIVE BUFFER, RECEIVE PACKET, RECEIVE RECORD または RECEIVE VARIABLEで行います。

ポート引数

最初の引数portで、ポートとプロトコルを指定します。

シリアルポートは99まで使用できます（一度に1つ）。以下の表はポートの値を示します。

ポートの値	説明
0	プリンタポート (Macintosh) または 無手順のCOM2 (PC)
1	モデムポート (Macintosh) または 無手順のCOM1 (PC)
20	プリンタポート (Macintosh) または XON/XOFFのようなソフトウェアフローのCOM2 (PC)
21	モデムポート (Macintosh) または XON/XOFFのようなソフトウェアフローのCOM1 (PC)
30	プリンタポート (Macintosh) または RTS/CTSのようなハードウェアフローのCOM2 (PC)
31	モデムポート (Macintosh) または RTS/CTSのようなハードウェアフローのCOM1 (PC)
101 to 199	無手順のシリアル通信
201 to 299	ソフトウェアフロー制御（XON/XOFF）のシリアルポート通信
301 to 399	ハードウェアフロー制御（RTS/CTS）のシリアルポート通信

重要: portに渡す値は、オペレーションシステムで認識される既存のシリアルCOMポートを示すものでなくてはなりません。例えば、101、103、125という値を使用できるようにするには、シリアルポートCOM1、COM3、COM25が必ず設定されている必要があります。

シリアルポートの注意点

標準の設定で、Mac OSとWindowsでは2つのシリアルポートが提供されています。Mac OSではモデムポートとプリンタポート、WindowsではCOM1とCOM2ポートです。しかしながら、シリアルポートはエクステンションボードを使用 して追加することができます。もともと4Dは2つの標準のシリアルポートのみサポートし、のちに追加されたポートをサポートしました。互換性のため、両方 をサポートするシステムとなっています。

- 標準のシリアルポート（プリンター/COM2またはモデム/COM1）にアクセスするには、port引数に0、1、20、21、30、31のいずれかを指定（以前の使用法に相当）するか、100以上の値にします（以下の説明を参照）。
- 追加したシリアルポートにアクセスするには、値をN+100にする必要があります（Nはポート番号）。ソフトウェアフローまたはハードウェアフローを選択するために、先の値（N+100）に100または200を加えることも考慮します。

例題 1

無手順のプリンタ/COM2ポートを使用する場合は、以下のシンタックスの1つを使用します:

```
SET CHANNEL(0;param)
```

または

```
SET CHANNEL(102;param)
```

例題 2

ソフトウェアフロー制御（XON/XOFF）のモデム/COM1ポートを使用する場合は、以下のシンタックスを使う必要があります:

```
SET CHANNEL(21;param)
```

または

```
SET CHANNEL(201;param)
```

例題 3

ハードウェアフロー制御（RTS/CTS等）のCOM25を使用する場合は、以下のシンタックスを使用する必要があります:

```
SET CHANNEL(325;param)
```

settings 引数

settings引数は転送速度、データビット、ストップビット、パリティを指定します。以下の表に示した転送速度、データビット、ストップビット、パリティの値を加算してsettingsの値を指定します。例えば、転送速度を1200ボー、データビットを8ビット、ストップビットを1、パリティをなし、と設定する場合には、94 + 3072 + 16384 + 0 = 19550と計算します。引数settingsの値として19550を指定します。

	settings 引数に加算する値	説明
転送速度	380	300
(ボー単位)	189	600
	94	1200
	62	1800
	46	2400
	30	3600
	22	4800
	14	7200
	10	9600
	4	19200
	2	28800
	1	38400
	0	57600
	1022	115200
	1021	230400
データビット	0	5
	2048	6
	1024	7
	3072	8
ストップビット	16384	1
	-32768	1.5
	-16384	2
パリティ	0	なし
	4096	奇数
	12288	偶数

Tip: port と settings に計算され渡されるさまざまな数値は、COM1...COM99の値を除き、テーマの定義済み変数で提供されています。COM1...COM99はリテラル数値を使用してください。

ディスク上のドキュメントを操作 - SET CHANNEL(operation;document)

SET CHANNELコマンドの第2の形式は、ドキュメントファイルの作成やオープンおよび、クローズを行います。コマンドと異なり、同時に1つのドキュメントファイルしか開くことができません。ドキュメントファイルは、読み込みと書き込みの両方が可能です。

operation引数で、document引数で指定したドキュメントに対する処理を指定します。以下の表に、operation の値と、その結果としてdocumentに対して行われる処理を示します。一番目の列はoperationで利用可能な値、二番目の列はdocumentで利用可能な値、三番目の列は処理を示します。

例えば、ファイルを開くダイアログボックスを表示して、テキストファイルを開く場合には、以下のように記述します:

```
SET CHANNEL(13;"")
```

Operation	Document	結果
10	文字列	文字列で指定されたドキュメントを開く。ドキュメントが存在しない場合、作成されます。
10	"" (空の文字列)	ファイルを開くダイアログを表示しファイルを開く。すべてのファイルタイプが表示されます。
11	なし	開かれたファイルを閉じる。
12	"" (空の文字列)	ファイルを保存ダイアログを表示し新しいファイルを作成。
13	"" (空の文字列)	ファイルを開くダイアログを表示しファイルを開く。テキストファイルのみが表示されます。

この表に示した処理はすべて、適切な場合、システム変数Documentに値を設定します。また、処理が正常に行われると、システム変数OKに1が代入されます。それ以外の場合には0が代入されます。

例題 4

RECEIVE BUFFER, SET TIMEOUT そして RECEIVE RECORDの例題を参照。

SET TIMEOUT

SET TIMEOUT (seconds)

引数	型		説明
seconds	倍長整数	→	タイムアウト秒数

説明

SET TIMEOUT は、シリアルポートコマンドの許容される待ち時間を設定します。シリアルポートコマンドが指定した時間seconds以内に終了しないと、そのシリアルポートコマンドは取り消され、エラー-9990が生成され、システム変数OKに0が代入されます。ON ERR CALLコマンドでインストールされるエラー処理メソッドを使用して、エラーをとらえることができます。

待ち時間はコマンドの実行に与えられたトータル時間です。データの送受信に要する時間ではない点に注意してください。以前の設定値を取り消し、シリアルポート通信のモニタを停止するには、secondsに0を指定します。

タイムアウトの設定が影響するコマンドは以下のとおりです:

- RECEIVE PACKET
- RECEIVE RECORD
- RECEIVE VARIABLE

例題

以下の例はシリアルポートからデータを受信します。タイムアウトを設定し、RECEIVE PACKETでデータを受け取ります。設定した時間内にデータを受け取れない場合、エラーが発生します:

```
SET CHANNEL(MacOS Serial Port;Speed 9600+Data bits 8+Stop bits One+Parity None) ` シリアルポートを開く
SET TIMEOUT(10) ` タイムアウトを10秒に設定
ON ERR CALL("CATCH COM ERRORS") ` メソッドが中断されないようにする
RECEIVE PACKET(vtBuffer;Char(13)) ` 改行まで読み込み
If(OK=0)
   ALERT("Error receiving data.")
Else
   [People]Name:=vtBuffer ` 受信したデータをフィールドに保存
End if
ON ERR CALL("")
```

USE CHARACTER SET

USE CHARACTER SET (map | * {; mapInOut})

引数	型		説明
map | *	文字, 演算子	→	使用する文字セット名 (Unicodeモード), または 使用するASCIIマップドキュメント名 (ASCIIモード), または * の場合、デフォルトの文字セット/ASCIIマップにリセット
mapInOut	倍長整数	→	0 = 出力マップ 1 = 入力マップ 省略した場合、出力マップ

説明

USE CHARACTER SETは、データベースとドキュメントまたはシリアルポート間のデータ交換の際、4Dが 使用する文字セットを変更するために使用できます。このコマンドはカレントプロセスに有効です。交換操作にはテキスト, DIF, そして SYLK ファイルの読み込み/書き出しが含まれます。文字マップはまたSEND PACKET, RECEIVE PACKET (テキスト型パケット) そしてRECEIVE BUFFERによるデータ交換にも使用されます。SEND RECORD, SEND VARIABLE, RECEIVE RECORD, SEND PACKET, RECEIVE PACKET (BLOB型パケット) そして RECEIVE VARIABLE によるデータ交換には影響しません。

map 引数は使用する文字セットの"IANA"名、またはそのエイリアス名と対応していなければなりません。例えば、"iso-8859-1"と"utf-8"は有効 な名前であり、そのエイリアス"latin1"あるいは"l1"もまた有効です。これらの名前に関する詳細は、以下のサイトを参照してください:

http://www.iana.org/assignments/character-sets

IANA名に関する説明はCONVERT FROM TEXTコマンドにもあります。

mapInOutが0の場合、マップは書き出しのために設定されます。mapInOutが1の場合、マップは読み込みのために設定されます。mapInOut 引数を渡さない場合、デフォルトで書き出しマップが使用されます。

* 引数が渡されると、(mapInOutの値に基づき、読み込みまたは書き込みが) デフォルトの文字セットに再設定されます。
4Dが使用するデフォルトの文字コードはUTF-8です。

例題

以下の例題 (Unicodeモード) では、UTF-16文字セットをテキストの書き出しに使用し、その後デフォルトの文字セットに再設定します:

```
USE CHARACTER SET("UTF-16LE";0) ` UTF-16 'リトルエンディアン' 文字セットを使用
EXPORT TEXT([MyTable];"MyText") ` マップを通してデータを書き出し
USE CHARACTER SET(*;0) ` デフォルト文字セットに戻す
```

システム変数およびセット

マップが正しくロードされるとシステム変数OKに1が設定され、そうでなければ0が設定されます。

配列

配列
配列を作成する
配列とフォームオブジェクト
配列と4D言語
配列とポインタ
配列の要素ゼロを使用する
二次元配列
配列とメモリ
APPEND TO ARRAY
ARRAY BLOB New 14.0
ARRAY BOOLEAN
ARRAY DATE
ARRAY INTEGER
ARRAY LONGINT
ARRAY OBJECT New 14.0
ARRAY PICTURE
ARRAY POINTER
ARRAY REAL
ARRAY TEXT
ARRAY TIME New 14.0
ARRAY TO LIST Updated 14.0
ARRAY TO SELECTION
BOOLEAN ARRAY FROM SET
COPY ARRAY
Count in array
DELETE FROM ARRAY
DISTINCT VALUES
Find in array
INSERT IN ARRAY
LIST TO ARRAY Updated 14.0
LONGINT ARRAY FROM SELECTION
MULTI SORT ARRAY
SELECTION RANGE TO ARRAY
SELECTION TO ARRAY
Size of array
SORT ARRAY
TEXT TO ARRAY
ARRAY STRING

配列

配列とは、同じタイプの変数を番号付きで並べたものです。各変数は、配列の**要素**といいます。配列の**サイズ**とは、配列が持つ要素の数を指します。配列は作成時にサイズが与えられ、要素の追加、挿入、削除によって、またはそれを作成するのに使用したのと同じコマンドを使用して、何度でもサイズを変更することができます。

配列は、配列宣言コマンドのいずれかを使用して作成します。詳細について配列を作成するを参照してください。

要素には、**1からN**の番号が付けられ、Nは配列のサイズとなります。配列には、常に**要素ゼロ**があります。この要素は他の要素と同様にアクセスできますが、配列をフォームに表示しても表示されません。配列がサポートするフォームオブジェクトには表示されませんが、プログラミング言語で要素ゼロを使用することに制限はありません。要素ゼロについての詳細は配列の要素ゼロを使用するを参照してください。

配列は4Dの変数です。他の変数と同様、配列にもスコープがあり、4D言語の規則に従いますが、他と異なるところがいくつかあります。詳細は配列と4D言語と配列とポインタを参照してください。

配列は言語上のオブジェクトですので、画面に表示されない配列を作成、使用できます。配列はまた、ユーザインタフェースオブジェクトでもあります。配列とフォームオブジェクトの間の相互作用についての詳細は配列とフォームオブジェクトとを参照してください。

配列は短時間に妥当な量のデータを保持するように設計されています。配列はメモリ内に保持されるため、取り扱いやすく、高速に操作できます。詳細については配列とメモリを参照してください。

配列を作成する

配列は、この章で説明する配列宣言コマンドのいずれかを使用して作成します。次の表に、配列宣言コマンドを記載します:

コマンド	配列の作成またはサイズ変更
ARRAY INTEGER	2バイト整数値
ARRAY LONGINT	4バイト整数値
ARRAY REAL	実数値
ARRAY TEXT	テキスト値(1要素は、2GBまでのテキスト) (*)
ARRAY STRING	テキスト値(廃止) (*)
ARRAY DATE	日付値
ARRAY BOOLEAN	ブール値
ARRAY PICTURE	ピクチャ値
ARRAY POINTER	ポインタ値
ARRAY OBJECT	ランゲージオブジェクト
ARRAY BLOB	BLOB
ARRAY TIME	時間

配列宣言コマンドは、1次元または2次元の配列の作成やサイズ変更が可能です。2次元配列の詳細は二次元配列を参照してください。

(*) テキスト配列と文字列配列には違いはありません。ARRAY STRING コマンドの strLen 引数は無視されます。テキスト配列の使用が推奨されます。ARRAY STRING コマンドは互換性のためだけに残されています。

次のコードは、10個の要素からなる整数配列を作成(宣言)します:

```
ARRAY INTEGER(aiAnArray;10)
```

次のコードは、同じ配列の要素を20個にサイズ変更します:

```
ARRAY INTEGER(aiAnArray;20)
```

次のコードは、同じ配列の要素をゼロにサイズ変更します:

```
ARRAY INTEGER(aiAnArray;0)
```

配列中の要素は中カッコ ({…}) を使用して参照します。中カッコの中には数字を入れて特定の要素を指定します。この数字を**要素番号**といいます。次の行は、5つの名前を atNames という配列に入れ、それらを警告ウィンドウに表示します:

```
ARRAY TEXT(atNames;5)
atNames{1}:="Richard"
atNames{2}:="Sarah"
atNames{3}:="Sam"
atNames{4}:="Jane"
atNames{5}:="John"
For($vlElem;1;5)
   ALERT("The element #"+String($vlElem)+" is equal to: "+atNames{$vlElem})
End for
```

atNames{$vlElem} というシンタックスに注目してください。atNames{3} のような数値そのものを指定するのではなく、数値変数を使用して配列の要素を指定できます。

ループ構造による反復を使用すると (For...End for, Repeat...Until または While...End while)、短いコードで配列のすべて、または一部のコードを指定することができます。

4D言語の配列関連コマンドなど

この他にも配列の作成や処理を行う4Dコマンドがあります。特に次のようなコマンドです:

- 配列とレコードセレクションを操作するにはSELECTION RANGE TO ARRAY, SELECTION TO ARRAY, ARRAY TO SELECTION そして DISTINCT VALUESコマンドを使用します。
- リストボックスタイプのオブジェクトは、配列に基づいています。"リストボックス"テーマのいくつかのコマンド（例えばLISTBOX INSERT ROWS)は配列とともに動作します。
- テーブルおよび配列に格納された値を元にグラフや図を作成できます。詳細はGRAPHコマンドを参照してください。
- LIST TO ARRAY と ARRAY TO LIST コマンド。
- 多 くのコマンドが、呼び出されると配列を構築します。例えば: FONT LIST, WINDOW LIST, VOLUME LIST, FOLDER LIST, DOCUMENT LIST, GET SERIAL PORT MAPPING, SAX GET XML ELEMENTなど。

配列とフォームオブジェクト

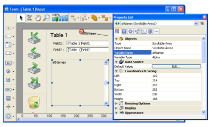
配列は言語オブジェクトです。決して画面に表示されない配列を作成、使用できます。しかし、配列は同時にユーザインタフェースオブジェクトでもあります。配列では、以下のタイプの**フォームオブジェクト**がサポートされます。

- ポップアップメニュー／ドロップダウンリスト
- コンボボックス
- スクロールエリア (4D v13より廃止予定)
- タブコントロール
- リストボックス

(リストボックスを除く)これらのオブジェクトは、デザインモードのフォームエディタで、プロパティリストウィンドウのデフォルト値ボタンを用いて事前に内容を定義できますが、配列コマンドを使用してプログラムで定義することもできます。いずれの場合も、フォームオブジェクトは開発者または4Dが作成した配列によってサポートされます。

これらのオブジェクトを使用するとき、その配列で**選択された要素**を調べることにより、オブジェクト内のどの項目が選択されているかを検出できます。逆に、選択する要素を配列に設定して、オブジェクトの中の特定の項目を選択することもできます。

配列がフォームオブジェクトのサポートに使用される場合、この配列は、ランゲージオブジェクトとユーザインタフェースの2つの性質を併せ持ちます。例えば、フォームにスクロールエリアを作成します:



Type	Scrollable Area
Object Name	Scrollable Area1
Variable Name	atNames
Variable Type	Alpha
Default Values	Edit...
Left	110
Top	114
Right	318
Bottom	282
Width	208
Height	168

関連付けられた変数の名前(この例ではatNames)が、スクロールエリアの作成と処理に使用する配列名となります。

Notes:

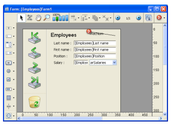
- 2次元配列やポインタ配列は表示できません。
- **リストボックス**タイプのオブジェクト(複数の配列を含む場合があります)を管理する上で、特定の状況が数々存在します。詳細についてはの節で説明しています。

例題: ドロップダウンリストを作成する

次の例では、配列に値を格納し、それをドロップダウンリストに表示する方法を示します。配列arSalariesは、ARRAY REALコマンドを使用して作成します。これには、社内で社員に支払われる基準給与がすべて入っています。ユーザがドロップダウンリストから要素を選択すると、[Employees]Salaryフィールドに選択された値が代入されます。

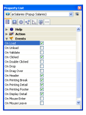
フォームにarSalariesドロップダウンリストを作成する

ドロップダウンリストを作成し、それをarSalariesと名付けます。ドロップダウンリストの変数名は、配列の名前と同じでなければなりません。


Form: [Employees]Form1
Employees
Last name : [Employees]Last name
First name : [Employees]First name
Position : [Employees]Position
Salary : [Employe arSalaries

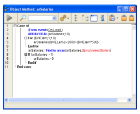
配列を初期化する

オブジェクトのOn Loadイベントを使用し、配列arSalariesを初期化します。これを行うには、次に示すように、**プロパティリスト**ウインドウでこのイベントを有効にします:



On Load	☑
On Unload	☐
On Validate	☐
On Clicked	☑
On Double Clicked	☐
On Drop	☐
On Drag Over	☐
On Header	☑
On Printing Break	☑
On Printing Detail	☑
On Printing Footer	☑
On Display Detail	☑
On Mouse Enter	☐
On Mouse Leave	☐

オブジェクトメソッドボタンをクリックして、次のようなメソッドを作成します:



```
Case of
      (Form event=On Load)
      ARRAY REAL(arSalaries;10)
   For ($VlElem;1;10)
         arSalaries($VlELem):=2000+($VlElem*500)
      End for
      arSalaries:=Find in array(arSalaries;[Employees]Salary)
   If (arSalaries=-1)
         arSalaries:=0
      End if
End case
```

以下のメソッドは:

```
ARRAY REAL(arSalaries;10)
For($vlElem;1;10)
   arSalaries{$vlElem}:=2000+($vlElem*500)
End for
```

税引き前の年間給与 ($30,000から$84,000まで) に対応する、2500, 3000... 7000の数値配列を作成します。

以下のメソッドは:

```
arSalaries:=Find in array(arSalaries;[Employees]Salary)
If(arSalaries=-1)
   arSalaries:=0
End if
```

新規レコードの作成と既存レコードの修正の両方を処理します。

- 新規レコードを作成する場合、[Employees]Salaryフィールドの初期値はゼロです。この場合、Find in array関数は配列の中で値を検索できず、-1を返します。If (arSalaries=-1) という判定式により、arSalariesをゼロに設定して、ドロップダウンリストで要素がまったく選択されていないことを示します。
- 既存レコードを修正する場合、Find in array関数により配列内の値の位置が取得され、ドロップダウンリスト要素を、フィールドの現在の値に対応するよう選択します。特定の従業員の値がリストにない場合、If (arSalaries=-1) という判定式により、リストの任意の要素の選択が解除されます。

Note: 配列選択要素についての詳細は、次の節を参照してください。

選択した値を[Employees]Salaryフィールドへ代入する

arSalariesドロップダウンリストで選択された値を調べるには、オブジェクトのOn Clickedイベントを処理する必要があります。選択された要素の要素番号は、配列arSalaries自体の値です。したがって、式arSalaries{arSalaries}はドロップダウンリストで選択した値を返します。

arSalaries ドロップダウンリストの完全なメソッドは、以下のようになります:

```
Case of
   :(Form event=On Load)
      ARRAY REAL(arSalaries;10)
      For($vlElem;1;10)
         arSalaries{$vlElem}:=2000+($vlElem*500)
      End for
      arSalaries:=Find in array(arSalaries;[Employees]Salary)
      If(arSalaries=-1)
         arSalaries:=0
      End if
   :(Form event=On Clicked)
      [Employees]Salary:=arSalaries{arSalaries}
End case
```

このドロップダウンリストは次のように表示されます。:

次の節では、配列をフォームオブジェクトとして使用する際の一般的な基本操作を説明します。

配列のサイズを取得する

配列の現在のサイズは、Size of arrayコマンドを使用して取得できます。前述の例の場合、次のコードは5を表示します:

```
ALERT("atNames配列のサイズは: "+String(Size of array(atNames)))
```

配列の要素を並べ替える

単一レベルの並び替えにはSORT ARRAYコマンドを、マルチレベルの並び替えにはMULTI SORT ARRAYコマンドを使用できます。配列がスクロールエリアに表示されているとします:

a. まずエリアは下図の左のリストのように表示されています。

b. 以下のコードを実行すると:

```
SORT ARRAY(atNames;>)
```

エリアは中央のように表示されます。

c. 以下のコードを実行すると:

```
SORT ARRAY(atNames;<)
```

tエリアは右のように表示されます。

要素の追加と削除

APPEND TO ARRAY, INSERT ELEMENTおよびDELETE ELEMENTコマンドを使用して、要素の追加、挿入、削除ができます。

配列内のクリックの処理: 選択した要素のテスト

前述の例で、配列がスクロールエリアに表示されているとき、このエリアでのクリックは、以下のように処理できます:

```
  ` atNames エリアオブジェクトメソッド
Case of
   :(Form event=On Load)
  ` 配列の初期化
      ARRAY TEXT(atNames;5)
  ` ...
   :(Form event=On Unload)
  ` 配列はもう必要ない
      CLEAR VARIABLE(atNames)

   :(Form event=On Clicked)
      If(atNames#0)
         vtInfo:="以下をクリックしました: "+atNames{atNames}
      End if

   :(Form event=On Double Clicked)
      If(atNames#0)
         ALERT("以下をダブルクリックしました: "+atNames{atNames}
      End if
End case
```

Note: 各イベントは、オブジェクトのプロパティでアクティブにしなければなりません。

シンタックスatNames{$vlElem}を使用して配列の特定の要素を処理でき、atNamesは配列中で選択されている要素の要素番号を返します。したがって、シンタックスatNames{atNames}は配列atNames内で選択されている要素の値を意味します。要素が選択されていない場合、atNamesはゼロになるため、If (atNames#0)という判定式は、要素が実際に選択されているかどうかを検出するために使用できます。

要素を選択済みに設定する

同様に、配列に値を割り当てることにより、プログラムから要素を選択できます。

配列内の値を検索する

```
  ` 一番目の要素を選択 (配列が空でなければ)
atNames:=1

  ` 最後の要素を選択 (配列が空でなければ)
atNames:=Size of array(atNames)

  ` 選択された要素を解除する
atNames:=0

If((0<atNames)&(atNames<Size of array(atNames))
  ` 可能であれば現在選択されている要素の次の要素を選択
   atNames:=atNames+1
End if

If(1<atNames)
  ` 可能であれば現在選択されている要素の前の要素を選択
   atNames:=atNames-1
End if
```

配列内の値を検索する

Find in arrayコマンドは、配列内の特定の値を検索します。前述の例を使用すると、次のコードは、リクエストダイアログボックスに"Richard"と入力した場合に、その値が"Richard"である要素を選択します:

```
$vsName:=Request("名前を入力:")
If(OK=1)
   $vlElem:=Find in array(atNames;$vsName)
   If($vlElem>0)
      atNames:=$vlElem
   Else
      ALERT($vsName+"は名前のリストに存在しません。")
   End if
End if
```

ポップアップメニュー、ドロップダウンリスト、スクロールエリア、タブコントロールは、通常同じ方法で扱うことができます。要素の値を変更したり、要素を追加や削除したりした場合でも、画面の再描画をするためにコードを追加する必要はありません。

Note: アイコン付きのタブコントロールを作成、使用したり、タブの有効と無効を切り替えるためには、タブコントロールをサポートするオブジェクトとして階層リストを使用しなければなりません。詳細はNew list コマンドの例題を参照してください。

コンボボックスの扱い

ポップアップメニュー、ドロップダウンリスト、スクロールエリア、タブコントロールは前節で説明したアルゴリズムによって制御できますが、コンボボックスの扱いはこれらと異なります。

コンボボックスは実際、値のリスト(配列の要素)が添付されたテキスト入力可能なエリアです。ユーザはこのリストから値を選択し、さらにテキストを編集できます。したがって、コンボボックスでは、選択した要素という概念は適用されません。

コンボボックスには、選択された要素というものはありません。ユーザがエリアに添付された値のいずれかを選択するたびに、その値が配列の要素ゼロに入ります。そしてユーザがテキストを編集すると、ユーザが変更した値が要素ゼロに入ります。

例題

```
  ` asColors コンボボックスオブジェクトメソッド
Case of
   :(Form event=On Load)
      ARRAY STRING(31;asColors;3)
      asColors{1}:="青"
      asColors{2}:="白"
      asColors{3}:="赤"
   :(Form event=On Clicked)
      If(asColors{0}#"")
  ` オブジェクトは自動でこの値を変更します
  ` コンボボックスのOn Clickedイベントは
  ` 追加の処理が必要な場合のみ使用されます
      End if
   :(Form event=On Data Change)
  ` Find in arrayは要素0を無視するので、-1または>0が返される
      If(Find in array(asColors;asColors{0})<0)
  ` 入力された値は、オブジェクトに添付された値の一つではない
  ` 再利用のためこの値をリストに追加する
         APPEND TO ARRAY(asColors;asColors{0})
      Else
  ` 入力された値は、リストに存在する
      End if
End case
```

配列と4D言語

配列は、4Dの変数です。他の変数と同様、配列にもスコープがあり、4D言語のルールに従いますが、いくつか他の変数とは異なる点があります。

ローカル、プロセス、インタープロセス配列

ローカル、プロセス、インタープロセス配列を作成して使用できます。次に例を示します:

```
ARRAY INTEGER($aiCodes;100) ` 100個の2バイト整数値でローカル配列を作成
ARRAY INTEGER(aiCodes;100) ` 100個の2バイト整数値でプロセス配列を作成
ARRAY INTEGER(<>aiCodes;100) ` 100個の2バイト整数値でインタープロセス配列を作成
```

これらの配列のスコープは、他のローカル、プロセス、インタープロセス変数のスコープと同じです。

ローカル配列

ローカル配列は、配列の名前をドル記号 ($) で始めることによって宣言されます。

ローカル配列のスコープは、それが作成されたメソッド内です。メソッドが終了すると配列も消去されます。2つの異なるメソッドにある同じ名前のローカル配列は、実際にはスコープの異なる2つの別の変数なため、それぞれに異なるタイプを指定できます。

フォームメソッド、オブジェクトメソッド、またはこれらのメソッドによってサブルーチンとして呼び出されるプロジェクトメソッドの中でローカル配列 を作成する場合、配列はフォームメソッドまたはオブジェクトメソッドが起動されるたびに作成され消去されます。つまり、配列はフォームイベントごとに作成 され消去されます。したがって、目的が表示であれ印刷であれ、ローカル配列をフォームに使用することはできません。

ローカル変数のように、可能な限りローカル配列を使用することをお勧めします。そうすることにより、多くの場合、アプリケーションを実行するために必要なメモリの量を小さくできます。

プロセス配列

プロセス配列は、配列名を文字で始めることによって宣言されます。

プロセス配列のスコープは、それが作成されたプロセス内です。配列は、プロセスの終了時またはアボート時に消去されます。プロセス配列には、プロセスごとに1つのインスタンスが自動的に作成されます。したがって、配列はプロセスのどこでも同じタイプです。ただし、その内容はプロセスによって異なります。

インタープロセス配列

インタープロセス配列は、配列名を<>で始めることによって宣言されます。

インタープロセス配列は、データベース全体で使用することができ、すべてのプロセスで共用されます。これらの使用は、プロセス間でのデータの共有や情報の転送のみに限る必要があります。

Tip: インタープロセス配列が複数のプロセスによってアクセスされ、それが衝突するおそれがあると予測できる場合、その配列へのアクセスをセマフォによって保護します。詳細については、Semaphore 関数の例題を参照してください。

Note: プロセス配列とインタープロセス配列をフォームの中で使用して、スクロールエリアやドロップダウンリスト等のフォームオブジェクトを作成できます。

配列を引数として受け渡し

配列を引数として4Dコマンドや4Dプラグインのルーチンに渡すことができます。しかし、配列をパラメータとしてメソッドに渡すことはできません。代わりに、配列へのポインタを引数としてメソッドに渡すことはできます。詳細については、を参照してください。

配列の他の配列への代入

テキストや文字列変数とは異なり、配列を他の配列に代入することはできません。配列を他の配列にコピー（代入）するには、COPY ARRAYコマンドを使用します。

配列とポインタ

配列を引数として4Dコマンドや4Dプラグインのルーチンに渡すことができます。しかし、配列を引数としてメソッドに渡すことはできません。代わりに、配列へのポインタを引数としてメソッドに渡すことはできます。

インタープロセス、プロセス、およびローカル配列のポインタを引数として渡すことができます。

次にいくつかの例を示します。

以下の例で:

```
If((0<atNames)&(atNames<Size of array(atNames))
 ` 可能であれば、選択された要素の次の要素を選択
   atNames:=atNames+1
End if
```

異なるフォームの50の異なる配列に同じ処理を実行する場合、次のプロジェクトメソッドを使用すると50回同じことを書かずにすみます:

```
 ` SELECT NEXT ELEMENT プロジェクトメソッド
 ` SELECT NEXT ELEMENT ( Pointer )
 ` SELECT NEXT ELEMENT ( -> Array )

C_POINTER($1)

If((0<$1->)&($1-><Size of array($1->))
   $1->:=$1->+1 ` 可能であれば、選択された要素の次の要素を選択
End if
```

これは以下のように使用します:

```
SELECT NEXT ELEMENT(->atNames)
 ` ...
SELECT NEXT ELEMENT(->asZipCodes)
 ` ...
SELECT NEXT ELEMENT(->alRecordIDs)
 ` ...
```

次のプロジェクトメソッドは、数値配列（整数、倍長整数または実数）のすべての要素の合計を返します:

```
 ` Array sum
 ` Array sum ( Pointer )
 ` Array sum ( -> Array )

C_REAL($0)

$0:=0
For($vlElem;1;Size of array($1->))
   $0:=$0+$1->{$vlElem}
End for
```

これは以下のように使用します:

```
$vlSum:=Array sum(->arSalaries)
 ` ...
$vlSum:=Array sum(->aiDefectCounts)
 ` ..
$vlSum:=Array sum(->alPopulations)
```

次のプロジェクトメソッドは、文字列またはテキスト配列のすべての要素を英大文字に変換します:

```
 ` CAPITALIZE ARRAY
 ` CAPITALIZE ARRAY ( Pointer )
 ` CAPITALIZE ARRAY ( -> Array )

For($vlElem;1;Size of array($1->))
   If($1->{$vlElem}#"")
      $1->{$vlElem}:=Uppercase($1->{$vlElem}[[1]])+Lowercase(Substring($1->{$vlElem};2))
   End if
End for
```

これは以下のように使用します:

```
CAPITALIZE ARRAY(->atSubjects)
 ` ...
CAPITALIZE ARRAY(->asLastNames)
```

配列、ポインタ、およびループ構造（等）を組み合わせることにより、配列を扱うための短くて便利なプロジェクトメソッドを作成できます。

配列の要素ゼロを使用する

配列には常に要素ゼロがあります。配列がフォームオブジェクトをサポートする場合、要素ゼロは表示されませんが、ランゲージでの使用に制限はありません。

(ひとつだけ例外があります。配列タイプのリストボックスでは、編集中の元の値を保持するため、内部的に0番目の要素が使用されています。この特別なケースでは、開発者は0番目の要素を使用できません。)

要素ゼロの用途の一例として、配列とフォームオブジェクトで説明しているコンボボックスの例があります。

その他の例を2つ挙げます。

1. 前に選択した要素以外の要素をクリックした場合のみ動作を行わせる場合、選択したそれぞれの要素を追跡する必要があります。これを実行する方法の1つは、選択する要素の要素番号を保持するプロセス変数を使用することです。もう1つの方法は、次のように配列の要素ゼロを使用する方法です:

```
  ` atNames スクロールエリアオブジェクトメソッド
 Case of
    :(Form event=On Load)
  ` 配列を初期化
       ARRAY TEXT(atNames;5)
  ` ...
  ` 要素番号0を、文字形式で、現在選択されている要素番号に初期化
  ` ここでは要素が選択されていないとする
       atNames{0}:="0"

    :(Form event=On Unload)
  ` もう配列は必要ない
       CLEAR VARIABLE(atNames)

    :(Form event=On Clicked)
       If(atNames#0)
          If(atNames#Num(atNames{0}))
             vtInfo:=atNames{atNames}+"がクリックされ、それは選択されていませんでした。"
             atNames{0}:=String(atNames)
          End if
       End if
    :(Form event=On Double Clicked)
       If(atNames#0)
          ALERT(atNames{atNames}+"がダブルクリックされました。")
       End if
 End case
```

(*) ただし、これには例外が一つあります。リストボックス型の配列では、要素ゼロは直前に編集された要素の値を保存するために内部的に使用されているので、この特定のコンテキストにおいては使用することはできません。

二次元配列

配列宣言コマンドはそれぞれ、1次元または2次元の配列を作成またはサイズ変更ができます。次に例を示します:

```
ARRAY TEXT(atTopics;100;50) ` 100行と50列からなるテキスト配列を作成
```

2次元配列は、本質的にはランゲージオブジェクトなため、表示も印刷もできません。

この例で、

- atTopicsは、2次元配列です。
- atTopics{8}{5}は、8行5列目の要素です。
- atTopics{20}は、20行目でそれ自身は1次元配列です。
- Size of array(atTopics)は、行数の100を返します。
- Size of array(atTopics{17})は、17行目の列数である50を返します。

以下の例では、データベースの各テーブルの各フィールドへのポインタが2次元配列に格納されます:

```
C_LONGINT($vlLastTable;$vlLastField)
C_LONGINT($vlFieldNumber)
  ` テーブルと同じ数の行 (空で、列なし) を作成
$vlLastTable:=Get last table number
ARRAY POINTER(<>apFields;$vlLastTable;0) `X行0列の2D配列
  ` テーブル毎に
For($vlTable;1;$vlLastTable)
   If(Is table number valid($vlTable))
      $vlLastField:=Get last field number($vlTable)
  ` 要素値を与える
      $vlColumnNumber:=0
      For($vlField;1;$vlLastField)
         If(Is field number valid($vlTable;$vlField))
            $vlColumnNumber:=$vlColumnNumber+1
  `テーブル行に列を挿入
            INSERT IN ARRAY(<>apFields{$vlTable};$vlColumnNumber;1)
  `セルにポインタを代入
            <>apFields{$vlTable}{$vlColumnNumber}:=Field($vlTable;$vlField)
         End if
      End for
   End if
End for
```

この2次元配列が初期化されたあと、以下の方法で特定のテーブルのフィールドへのポインタを取得できます:

```
  ` 画面に現在表示されているテーブルの、フィールドへのポインタを取得：
COPY ARRAY(<>apFields{Table(Current form table)};$apTheFieldsIamWorkingOn)
  ` ブールと日付フィールドを初期化
For($vlElem;1;Size of array($apTheFieldsIamWorkingOn))
   Case of
      :(Type($apTheFieldsIamWorkingOn{$vlElem}->)=Is Date)
         $apTheFieldsIamWorkingOn{$vlElem}->:=Current date
      :(Type($apTheFieldsIamWorkingOn{$vlElem}->)=Is Boolean)
         $apTheFieldsIamWorkingOn{$vlElem}->:=True
   End case
End for
```

Note: この例でわかるように、2次元配列の行のそれぞれの列数は同じサイズでも異なるサイズでも構いません。

配列とメモリ

テーブルやレコードを使用してディスク上に格納したデータと異なり、配列は常に全部がメモリに保持されます。

例えば、米国内の郵便番号がすべて[Zip Codes]テーブルに入力されている場合、約100,000件のレコードになります。加えて、そのテーブルは郵便番号の他に、対応する市、郡、州という複数のフィールドを持つとします。カリフォルニアからのみ郵便番号を選択する場合、4Dデータベースエンジンが[Zip Codes]テーブルの対応するレコードセレクションを作成して、必要な場合にのみそのレコードをロードします（例えば表示や印刷時）。すなわち、4Dのデータベースエンジンによってディスクからメモリに部分的にロードされた（フィールドごとに同じタイプの）順序づけられた一連の値で作業するということです。

同じことを配列で実行するのは、次の理由で禁止されています:

- 4つの情報タイプ（郵便番号、市、郡、州）を維持するためには、4つの大きな配列をメモリ内で維持する必要があります。
- 配列は、常に全体がメモリ内に維持されるため、あまり使用しない場合でも、作業セッションの間すべての郵便番号をメモリに置いておく必要があります。
- 同じく配列全体が常にメモリ内に維持されることから、データが作業セッション中に使用も変更もされていない場合でも、4つの配列をデータベースが開始されるたびディスクからロードして、終了時にはディスクに保存する必要があります。

結論: 配列は、ほどよい量のデータを短時間維持するためのものです。他方、配列はメモリ内に置かれるため、扱いやすく高速操作が可能です。

しかし、状況によっては何百、何千という要素を持った配列で作業する必要があります。
次の表に、各配列がメモリ上に占めるバイト数を求めるための計算式を示します:

配列型	メモリ使用量の計算式 (バイト単位)
ブール	(31+要素数)\8
日付	(1+要素数) * 6
文字列	(1+要素数) * (各テキストの合計サイズ)
整数	(1+要素数) * 2
倍長整数	(1+要素数) * 4
ピクチャ	(1+要素数) * 4 + ピクチャサイズの合計
ポインタ	(1+要素数) * 16
実数	(1+要素数) * 8
テキスト	(1+要素数) * (各テキストの合計サイズ)
二次元	(1+要素数) * 12 + 配列サイズの合計

注:

- メモリ中のテキストサイズは以下の式で計算されます: ((Length + 1) * 2)
- 選択した要素、要素数、配列自体の情報を保持するため、さらに数バイトを要します。

非常に大きな配列で作業する場合、メモリが一杯という状況を扱う最善の方法は、配列の作成をON ERR CALLで囲むことです。以下に例を示します:

```
  ` 大きな配列を作成する必要がある晩に一晩かけてバッチ操作を実行します。
  ` 夜間にエラーが発生する危険がないように、操作の始めに配列を作成して、
  ` この時点でエラーをテストします。
gError:=0 ` エラーなし状態にセット
ON ERR CALL("ERROR HANDLING") ` エラーをキャッチするメソッドをインストール
ARRAY STRING(63;asThisArray;50000) ` 非Unicodeモードで約3125K
ARRAY REAL(arThisAnotherArray;50000) ` 488K
ON ERR CALL("") ` これ以降は、エラー処理不要
If(gError=0)
  ` 配列を作ることができた
  ` そして、操作を続行できる
Else
   ALERT("この操作はメモリ不足により続行ができません！")
End if
  ` これ以降は、配列は不要となる
CLEAR VARIABLE(asThisArray)
CLEAR VARIABLE(arThisAnotherArray)
```

ERROR HANDLING プロジェクトメソッドは以下のとおりです:

```
  ` ERROR HANDLING プロジェクトメソッド
gError:=Error ` エラーコードを返す
```

APPEND TO ARRAY

APPEND TO ARRAY (array ; value)

引数	型	説明
array	配列	⇒ 要素を追加する配列
value	式	⇒ 追加する値

説明

APPEND TO ARRAYコマンドは、arrayの最後に新規要素を追加し、その要素にvalueを代入します。インタプリタモードでは、arrayが存在しない場合、コマンドはvalueの型に対応する配列を作成します。

このコマンドはあらゆるタイプの配列（文字列、数値、ブール、日付、ポインタ、ピクチャ）に対応します。

valueのタイプは配列のタイプと一致しなくてはなりません。一致しない場合は、シンタックスエラー54"引数のタイプが違います"が生成されます。ただし、次の値は受け入れられます。

- 文字列配列（テキストまたは文字列）はテキストや文字列タイプのvalueを受け入れます。
- 数値配列（整数、倍長整数、実数）は、整数、倍長整数、実数、または時間タイプのvalueを受け入れます。

例題

以下のコードは:

```
INSERT IN ARRAY($myarray;Size of array($myarray)+1)
$myarray{Size of array($myarray)}:=$myvalue
```

このコードに置き換えることができます:

```
APPEND TO ARRAY($myarray;$myvalue)
```

ARRAY BLOB

ARRAY BLOB (arrayName ; size {; size2})

引数	型		説明
arrayName	配列	→	配列名
size	倍長整数	→	配列の要素の数、もしくはsize2が指定されていた場合は配列の数
size2	倍長整数	→	2次元配列の要素の数

説明

ARRAY BLOBコマンドは、メモリ上にBLOB型の要素の配列を作成・リサイズします。

arrayName 引数には配列の名前を渡します。

size 引数には、配列の要素の数を渡します。

size2 引数は任意の引数です。指定時にはコマンドは2次元配列を作成します。この場合、size 引数はそれぞれの配列の行の数を指定し、size2 引数はそれぞれの配列の列の数を指定します。2次元配列内のそれぞれの行は、要素としても配列としても扱うことが出来ます。これはつまり、配列の1次元目を扱うときは、このテーマ内の他のコマンドを使用することによって2次元配列の中に配列全体を挿入したり削除したりすることが出来るということです。

ARRAY BLOB コマンドを既存の配列に対して使用する場合、以下のことに気を付けてください:

- サイズを拡大する場合、既存の要素は何も変更されず、新しく追加された要素は空のBLOBで初期化されます(BLOB size= 0)
- サイズを縮小する場合は、"底"にある要素から削除されていきます。

例題 1

以下の例はBLOB型の要素を100個含んだプロセス配列を作成します:

```
ARRAY BLOB(arrBlob;100)
```

例題 2

以下の例は50個のBLOB型の要素を含んだ行を100行持ったローカルな配列を作成します:

```
ARRAY BLOB($arrBlob;100;50)
```

例題 3

以下の例は50個のBLOB型の要素を含んだ行を100行持ったローカルな配列を作成します。$vByteValue 変数には、そのBLOB配列の5行目・7列目の、10バイト目のBLOBが渡されます:

```
C_INTEGER($vByteValue)
ARRAY BLOB($arrValues;100;50)
...
$vByteValue:=$arrValues{5}{7}{9}
```

ARRAY BOOLEAN

ARRAY BOOLEAN (arrayName ; size {; size2})

引数	型		説明
arrayName	配列	→	配列名
size	倍長整数	→	配列の要素数、またはsize2を指定した場合は配列の行数
size2	倍長整数	→	2次元配列の列数

説明

ARRAY BOOLEANコマンドは、メモリ上にブール要素の配列を作成またはリサイズします。

arrayName引数は作成する配列の名前です。

size引数は配列の要素数です。

size2引数はオプションです。size2が渡されている場合、コマンドは2次元配列を作成します。この場合、sizeに配列の行数を、size2にはそれぞれの配列の列数を指定します。2次元配列の各行は要素および配列として扱えます。これは配列の一番目の次元を扱う時、2次元配列中の配列全体を挿入あるいは削除するために、他の配列コマンドを使用できることを意味します。

ARRAY BOOLEANを既存の配列に適用する場合、

- 配列サイズを増やす場合、既存の値は保持され、新しい要素はFalseで初期化されます。
- 配列サイズを減らす場合、後ろの要素は配列から削除され、失われます。

Tip: コンテキストによっては、ブール配列の代わりに整数配列を使用し、要素が非0値の場合はTrue、0の場合はFalseとすることもできます。

例題 1

この例は、100要素のﾌﾞｰﾙプロセス配列を作成します:

```
ARRAY BOOLEAN(abValues;100)
```

例題 2

この例は、100行50列要素のﾌﾞｰﾙローカル配列を作成します:

```
ARRAY BOOLEAN($abValues;100;50)
```

例題 3

この例は、50要素のﾌﾞｰﾙインタープロセス配列を作成し、それぞれの偶数要素にTrueを格納します:

```
ARRAY BOOLEAN(<>abValues;50)
For($vlElem;1;50)
   <>abValues{$vlElem}:=(($vlElem%2)=0)
End for
```

ARRAY DATE

ARRAY DATE (arrayName ; size {; size2})

引数	型		説明
arrayName	配列	→	配列名
size	倍長整数	→	配列の要素数、またはsize2を指定した場合は配列の行数
size2	倍長整数	→	2次元配列の列数

説明

ARRAY DATEコマンドは、メモリ上に日付要素の配列を作成またはリサイズします。

arrayName引数は作成する配列の名前です。

size引数は配列の要素数です。

size2引数はオプションです。size2が渡されている場合、コマンドは2次元配列を作成します。この場合、sizeに配列の行数を、size2にはそれぞれの配列の列数を指定します。2次元配列の各行は要素および配列として扱えます。これは配列の一番目の次元を扱う時、2次元配列中の配列全体を挿入あるいは削除するために、他の配列コマンドを使用できることを意味します。

ARRAY DATEを既存の配列に適用する場合、

- 配列サイズを増やす場合、既存の値は保持され、新しい要素は空の日付 (!00/00/00!) で初期化されます。
- 配列サイズを減らす場合、後ろの要素は配列から削除され、失われます。

例題 1

この例は、100要素の日付プロセス配列を作成します:

```
ARRAY DATE(adValues;100)
```

例題 2

この例は、100行50列要素の日付ローカル配列を作成します:

```
ARRAY DATE($adValues;100;50)
```

例題 3

この例は、50要素の日付インタープロセス配列を作成し、それぞれの要素に現在の日付+要素番号を格納します:

```
ARRAY DATE(<>adValues;50)
For($vlElem;1;50)
   <>adValues{$vlElem}:=Current date+$vlElem
End for
```

ARRAY INTEGER

ARRAY INTEGER (arrayName ; size {; size2})

引数	型	説明
arrayName	配列	→ 配列名
size	倍長整数	→ 配列の要素数、またはsize2を指定した場合は配列の行数
size2	倍長整数	→ 2次元配列の列数

説明

ARRAY INTEGERコマンドは、メモリ上に2バイト整数要素の配列を作成またはリサイズします。

arrayName引数は作成する配列の名前です。

size引数は配列の要素数です。

size2引数はオプションです。size2が渡されている場合、コマンドは2次元配列を作成します。この場合、sizeに配列の行数を、size2にはそれぞれの配列の列数を指定します。2次元配列の各行は要素および配列として扱えます。これは配列の一番目の次元を扱う時、2次元配列中の配列全体を挿入あるいは削除するために、他の配列コマンドを使用できることを意味します。

ARRAY INTEGERを既存の配列に適用する場合、

- 配列サイズを増やす場合、既存の値は保持され、新しい要素は0で初期化されます。
- 配列サイズを減らす場合、後ろの要素は配列から削除され、失われます。

例題 1

この例は、100要素の整数プロセス配列を作成します:

```
ARRAY INTEGER(aiValues;100)
```

例題 2

この例は、100行50列要素の整数ローカル配列を作成します:

```
ARRAY INTEGER($aiValues;100;50)
```

例題 3

この例は、50要素の整数インタープロセス配列を作成し、それぞれの要素に要素番号を格納します:

```
ARRAY INTEGER(<>aiValues;50)
For($vlElem;1;50)
   <>aiValues{$vlElem}:=$vlElem
End for
```

ARRAY LONGINT

ARRAY LONGINT (arrayName ; size {; size2})

引数	型		説明
arrayName	配列	→	配列名
size	倍長整数	→	配列の要素数、またはsize2を指定した場合は配列の行数
size2	倍長整数	→	2次元配列の列数

説明

ARRAY LONGINTコマンドは、メモリ上に4バイト倍長整数要素の配列を作成またはリサイズします。

arrayName引数は作成する配列の名前です。

size引数は配列の要素数です。

size2引数はオプションです。size2が渡されている場合、コマンドは2次元配列を作成します。この場合、sizeに配列の行数を、size2にはそれぞれの配列の列数を指定します。2次元配列の各行は要素および配列として扱えます。これは配列の一番目の次元を扱う時、2次元配列中の配列全体を挿入あるいは削除するために、他の配列コマンドを使用できることを意味します。

ARRAY LONGINTを既存の配列に適用する場合、

- 配列サイズを増やす場合、既存の値は保持され、新しい要素は0で初期化されます。
- 配列サイズを減らす場合、後ろの要素は配列から削除され、失われます。

例題 1

この例は、100要素の倍長整数プロセス配列を作成します:

```
ARRAY LONGINT(alValues;100)
```

例題 2

この例は、100行50列要素の倍長整数ローカル配列を作成します:

```
ARRAY LONGINT($alValues;100;50)
```

例題 3

この例は、50要素の倍長整数インタープロセス配列を作成し、それぞれの要素に要素番号を格納します:

```
ARRAY LONGINT(<>alValues;50)
For($vlElem;1;50)
   <>alValues{$vlElem}:=$vlElem
End for
```

ARRAY OBJECT

ARRAY OBJECT (arrayName ; size {; size2})

引数	型		説明
arrayName	配列	→	配列名
size	倍長整数	→	配列の要素の数、もしくはsize2が指定されていた場合は配列の数
size2	倍長整数	→	2次元配列の要素の数

説明

ARRAY OBJECTコマンドは、メモリ上にランゲージオブジェクト型の要素の配列を作成・リサイズします。

arrayName 引数には配列の名前を渡します。4Dのルールに則った名前であればどんな名前でも使用することができます。

size 引数には、配列の要素の数を渡します。

size2 引数は任意の引数です。指定時にはコマンドは2次元配列を作成します。この場合、size 引数はそれぞれの配列の行の数を指定し、size2 引数はそれぞれの配列の列の数を指定します。2次元配列内のそれぞれの行は、要素としても配列としても扱うことが出来ます。これはつまり、配列の1次元目を扱うときは、"配列"テーマ内の他のコマンドを使用することによって2次元配列の中に配列全体を挿入したり削除したりすることが出来るということです。

ARRAY OBJECT コマンドを既存の配列に対して使用する場合、以下のことに注意して下さい:

- サイズを拡大する場合、既存の要素は何も変更されず、新しく追加された要素は未定義の要素になります。OB Is defined コマンドを使用することによって要素が定義済みかどうかを調べることができます。
- サイズを縮小する場合は、"底"にある要素から削除されていきます。

例題 1

以下の例はオブジェクト型の要素を100個含んだプロセス配列を作成します:

```
ARRAY OBJECT(arrObjects;100)
```

例題 2

以下の例は50個のオブジェクト型の要素を含んだ行を100行持ったローカルな配列を作成します:

```
ARRAY OBJECT($arrObjects;100;50)
```

例題 3

以下の例はローカルなオブジェクト配列を作成してデータをそこに代入します:

```
C_OBJECT($Children;$ref_richard;$ref_susan;$ref_james)
ARRAY OBJECT($arrayChildren;0)
OB SET($ref_richard;"name";"Richard";"age";7)
APPEND TO ARRAY($arrayChildren;$ref_richard)
OB SET($ref_susan;"name";"Susan";"age";4)
APPEND TO ARRAY($arrayChildren;$ref_susan)
OB SET($ref_james;"name";"James";"age";3)
APPEND TO ARRAY($arrayChildren;$ref_james)
 // $arrayChildren{1} -> {"name":"Richard","age":7}
 // $arrayChildren{2} -> {"name":"Susan","age":4}
 // $arrayChildren{3} -> {"name":"James","age":3}
```

ARRAY PICTURE

ARRAY PICTURE (arrayName ; size {; size2})

引数	型		説明
arrayName	配列	→	配列名
size	倍長整数	→	配列の要素数、またはsize2を指定した場合は配列の行数
size2	倍長整数	→	2次元配列の列数

説明

ARRAY PICTUREコマンドは、メモリ上にピクチャ要素の配列を作成またはリサイズします。

arrayName引数は作成する配列の名前です。

size引数は配列の要素数です。

size2引数はオプションです。size2が渡されている場合、コマンドは2次元配列を作成します。この場合、sizeに配列の行数を、size2にはそれぞれの配列の列数を指定します。2次元配列の各行は要素および配列として扱えます。これは配列の一番目の次元を扱う時、2次元配列中の配列全体を挿入あるいは削除するために、他の配列コマンドを使用できることを意味します。

ARRAY PICTUREを既存の配列に適用する場合、

- 配列サイズを増やす場合、既存の値は保持され、新しい要素は空のピクチャで初期化されます。この新しい要素にPicture sizeを適用した場合、0が返されます。
- 配列サイズを減らす場合、後ろの要素は配列から削除され、失われます。

例題 1

この例は、100要素のﾋﾟｸﾁｬプロセス配列を作成します:

```
ARRAY PICTURE(agValues;100)
```

例題 2

この例は、100行50列要素のﾋﾟｸﾁｬローカル配列を作成します:

```
ARRAY PICTURE($agValues;100;50)
```

例題 3

この例題ではインタープロセスピクチャ配列を作成し、ピクチャをそれぞれの要素にロードします。配列の要素数は'PICT' リソース数と同じです。配列のリソース名は"User Intf/"から始まります。:

```
RESOURCE LIST("PICT";$aiResIDs;$asResNames)
ARRAY PICTURE(<>agValues;Size of array($aiResIDs))
$vlPictElem:=0
For($vlElem;1;Size of array(<>agValues))
   If($asResNames{$vlElem}="User Intf/@")
      $vlPictElem:=$vlPictElem+1
      GET PICTURE RESOURCE("PICT";$aiResIDs{$vlElem};$vgPicture)
      <>agValues{$vlPictElem}:=$vgPicture
   End if
End for
ARRAY PICTURE(<>agValues;$vlPictElem)
```

ARRAY POINTER

ARRAY POINTER (arrayName ; size {; size2})

引数	型	説明
arrayName	配列	→ 配列名
size	倍長整数	→ 配列の要素数、またはsize2を指定した場合は配列の行数
size2	倍長整数	→ 2次元配列の列数

説明

ARRAY POINTERコマンドは、メモリ上にポインタ要素の配列を作成またはリサイズします。

arrayName引数は作成する配列の名前です。

size引数は配列の要素数です。

size2引数はオプションです。size2が渡されている場合、コマンドは2次元配列を作成します。この場合、sizeに配列の行数を、size2にはそれぞれの配列の列数を指定します。2次元配列の各行は要素および配列として扱えます。これは配列の一番目の次元を扱う時、2次元配列中の配列全体を挿入あるいは削除するために、他の配列コマンドを使用できることを意味します。

ARRAY POINTERを既存の配列に適用する場合、

- 配列サイズを増やす場合、既存の値は保持され、新しい要素はヌルポインタで初期化されます。この新しい要素にNilを適用するとTrueが返されます。
- 配列サイズを減らす場合、後ろの要素は配列から削除され、失われます。

例題 1

この例は、100要素のポインタプロセス配列を作成します:

```
ARRAY POINTER(apValues;100)
```

例題 2

この例は、100行50列要素のポインタローカル配列を作成します:

```
ARRAY POINTER($apValues;100;50)
```

例題 3

この例題はポインタインタプロセス配列を作成し、それぞれの要素に要素番号に対応するテーブルのポインタを格納します。要素数はテーブル数と同じです。テーブルが削除されていた場合、Nilが返されます。

```
ARRAY POINTER(<>apValues;Get last table number)
For($vlElem;1;Size of array(<>apValues);1;-1)
   If(Is table number valid($vlElem))
      <>apValues{$vlElem}:=Table($vlElem)
   End if
End for
```

ARRAY REAL

ARRAY REAL (arrayName ; size {; size2})

引数	型		説明
arrayName	配列	→	配列名
size	倍長整数	→	配列の要素数、またはsize2を指定した場合は配列の行数
size2	倍長整数	→	2次元配列の列数

説明

ARRAY REALコマンドは、メモリ上に実数要素の配列を作成またはリサイズします。

arrayName引数は作成する配列の名前です。

size引数は配列の要素数です。

size2引数はオプションです。size2が渡されている場合、コマンドは2次元配列を作成します。この場合、sizeに配列の行数を、size2にはそれぞれの配列の列数を指定します。2次元配列の各行は要素および配列として扱えます。これは配列の一番目の次元を扱う時、2次元配列中の配列全体を挿入あるいは削除するために、他の配列コマンドを使用できることを意味します。

ARRAY REALを既存の配列に適用する場合、

- 配列サイズを増やす場合、既存の値は保持され、新しい要素は0で初期化されます。
- 配列サイズを減らす場合、後ろの要素は配列から削除され、失われます。

例題 1

この例は、100要素の実数プロセス配列を作成します:

```
ARRAY REAL(arValues;100)
```

例題 2

この例は、100行50列要素のブールローカル配列を作成します:

```
ARRAY REAL($arValues;100;50)
```

例題 3

この例は、50要素の実数インタープロセス配列を作成し、それぞれの要素に要素番号を格納します:

```
ARRAY REAL(<>arValues;50)
For($vlElem;1;50)
   <>arValues{$vlElem}:=$vlElem
End for
```

ARRAY TEXT

ARRAY TEXT (arrayName ; size {; size2})

引数	型	説明
arrayName	配列	→ 配列名
size	倍長整数	→ 配列の要素数、またはsize2を指定した場合は配列の行数
size2	倍長整数	→ 2次元配列の列数

説明

ARRAY TEXTコマンドは、メモリ上にテキスト要素の配列を作成またはリサイズします。

arrayName引数は作成する配列の名前です。

size引数は配列の要素数です。

size2引数はオプションです。size2が渡されている場合、コマンドは2次元配列を作成します。この場合、sizeに配列の行数を、size2にはそれぞれの配列の列数を指定します。2次元配列の各行は要素および配列として扱えます。これは配列の一番目の次元を扱う時、2次元配列中の配列全体を挿入あるいは削除するために、他の配列コマンドを使用できることを意味します。

ARRAY TEXTを既存の配列に適用する場合、

- 配列サイズを増やす場合、既存の値は保持され、新しい要素は空の文字 "" で初期化されます。
- 配列サイズを減らす場合、後ろの要素は配列から削除され、失われます。

例題 1

この例は、100要素のテキストプロセス配列を作成します:

```
ARRAY TEXT(atValues;100)
```

例題 2

この例は、100行50列要素のテキストローカル配列を作成します:

```
ARRAY TEXT($atValues;100;50)
```

例題 3

この例は、50要素のテキストインタープロセス配列を作成し、それぞれの要素に"Element #"および要素番号文字列を格納します:

```
ARRAY TEXT(<>atValues;50)
For($vlElem;1;50)
   <>atValues{$vlElem}:="Element #"+String($vlElem)
End for
```

ARRAY TIME

ARRAY TIME (arrayName ; size {; size2})

引数	型		説明
arrayName	配列	→	配列名
size	倍長整数	→	配列の要素の数、もしくはsize2が指定されていた場合は配列の数
size2	倍長整数	→	2次元配列の要素の数

説明

ARRAY TIMEコマンドは、メモリ上に時間型の要素の配列を作成・リサイズします。

リマインダ: 4Dでは時間は数の値として処理されるということに注意して下さい。以前のバージョンの4Dでは、時間の配列を管理するためには倍長整数配列と表示フォーマットを組み合わせる必要がありました。

arrayName 引数には配列の名前を渡します。

size 引数には、配列の要素の数を渡します。

size2 引数は任意の引数です。指定時にはコマンドは2次元配列を作成します。この場合、size 引数はそれぞれの配列の行の数を指定し、size2 引数はそれぞれの配列の列の数を指定します。2次元配列内のそれぞれの行は、要素としても配列としても扱うことが出来ます。これはつまり、配列の1次元目を扱うときは、このテーマ内の他のコマンドを使用することによって2次元配列の中に配列全体を挿入したり削除したりすることが出来るということです。

ARRAY TIME コマンドを既存の配列に対して使用するときには以下の点に注意して下さい:

- サイズを拡大する場合、既存の要素は何も変更されず、新しく追加された要素はnull時間の値(00:00:00)で初期化されます。
- サイズを縮小する場合は、"底"にある要素から削除されていきます。

SELECTION TO ARRAY または SELECTION RANGE TO ARRAY のコマンドを Time 型のフィールドに対して適用した場合、これらのコマンドはコピー先の配列が他の型(例えば倍長整数など)に定義されていない場合に限り、時間型の配列を作成します。

例題 1

以下の例は時間型の要素を100個含んだプロセス配列を作成します:

```
ARRAY TIME(arrTimes;100)
```

例題 2

以下の例は50個の時間型の要素を含んだ行を100行持ったローカルな配列を作成します:

```
ARRAY TIME($arrTimes;100;50)
```

例題 3

時間の配列が数字の値を受け取ることが出来るようになったことによって、以下のようなコードを使用することが出来るようになります:

```
ARRAY TIME($arrTimeValues;10)
$CurTime:=Current time+1
APPEND TO ARRAY($arrTimeValues;$CurTime)
$Found:=Find in array($arrTimeValues;$CurTime)
```

ARRAY TO LIST

ARRAY TO LIST (array ; list {; itemRefs})

引数	型		説明
array	配列	→	配列要素のコピー元配列
list	文字, ListRef	→	配列要素のコピー先リスト
itemRefs	配列	→	項目参照番号の数値配列

説明

ARRAY TO LIST コマンドは、配列arrayの要素を使用して、(デザインモードのリストエディタで作成される)階層リストまたは選択リストlistリストを作成または置き換えます。

list 引数には、選択リスト(文字列)または階層リスト参照(ListRef) を渡すことが出来ます。後者の場合、このコマンドが動くためには、渡すリストは(例えば New list コマンドを使用するなどして) 事前に作成されている必要があります。

任意の引数itemRefsが渡される場合、この配列は数値配列で、array配列と同期していなければなりません。各要素は、arrayの対応する要素のリスト項目参照番号を表わします。この引数を省略した場合、4Dにより自動的に1、2...Nという項目参照番号が設定されます。

互換性に関する注意: ARRAY TO LIST コマンドは、以下の様な制限があることに注意しなければなりません。

- このコマンドはリストの第1レベルの項目のみ設定することが出来ます。
- このコマンドを選択リストに使用すると、アプリケーションのストラクチャを変更する(リストはストラクチャファイルに保存されています)ので、製品のストラクチャファイルがアップデートされた際にローカルな変更は全て失われてしまいます。
- このコマンドはコンポーネント内で使用することはできません。なぜならコンポーネントの選択リストはストラクチャとともに読み出し専用としてロードされるからです。

ARRAY TO LIST コマンドを使い、配列の要素に基づくリストを作成することはできます。しかしながら、これらの制約に縛られずにリストの値を自由に使用するためには、階層リスト テーマ内にあるコマンドの使用が推奨されます。

例題

以下の例は、配列atRegionsの要素から構成される"Regions"リストを作成します:

```
ARRAY TO LIST(atRegions;"Regions")
```

例題

フィールドの異なる値をリストに入れて、例えば階層ポップアップメニューを作成したい場合:

```
ALL RECORDS([Company])
DISTINCT VALUES([Company]country;$arrCountries)
CountryList:=New list
ARRAY TO LIST($arrCountries;CountryList)
```

エラー管理

デザインモードのリストエディタで現在編集中のリストにARRAY TO LISTを適用すると、エラー -9957 が生成されます。このエラーはON ERR CALLでインストールされたエラー処理メソッドで管理できます。

ARRAY TO SELECTION

ARRAY TO SELECTION {(array ; aField {; array2 ; aField2 ; ... ; arrayN ; aFieldN}{; *})}

引数	型		説明
array	配列	→	コピー元の配列
aField	フィールド	←	配列データを受け取るフィールド
*	演算子	→	実行をスタックする

説明

ARRAY TO SELECTIONコマンドは、1つ以上の配列をレコードのセレクションにコピーします。すべてのフィールドは同一テーブルのものでなければなりません。

コマンド呼び出し時にセレクションが存在する場合、配列の並び順とレコードの並び順に基づき、配列要素はレコードに書き込まれます。要素数がレコード数よりも多い場合、新しいレコードを作成します。レコードは、既存でも新規でも、自動的に保存されます。

すべての配列は同じ要素数でなければなりません。配列のサイズが異なる場合、シンタックスエラーが生成されます。

このコマンドはSELECTION TO ARRAYコマンドとは逆の動作を行います。しかし**ARRAY TO SELECTION**コマンドは、たとえ自動リレートが設定されていても、リレートテーブルを含む他のテーブルのフィールドを使用することはできません。

* 引数を渡すと、4Dはその行の実行を遅延し、メモリに格納します。* で終わる行を使用して複数の行をスタックできます。スタックされた行は * なしの**ARRAY TO SELECTION**一回の呼び出しですべて実行されます。この目的のため、このコマンドを引数なしで呼び出すことができます。
これにより、QUERYコマンドのように、複雑な文を複数の行に分割して記述することができ、可読性が向上します。また途中の行の挿入も容易です。

警告: **ARRAY TO SELECTION**コマンドは、既存のレコードの情報を上書きします。十分に注意して使用してください。**ARRAY TO SELECTION**コマンド実行中、レコードが他のプロセスによりロックされていると、そのレコードは更新されません。ロックされたレコードは"**LockedSet**"というプロセスセットに入れられます。**ARRAY TO SELECTION**コマンド実行後に"**LockedSet**"セットをテストして、ロックされていたレコードの存在を知ることができます。

4D Server: このコマンドは4D Server用に最適化されています。配列はクライアントマシンからサーバへ渡され、レコードの修正や追加はサーバ上で実行されます。この処理は同期的に 行われるため、クライアントマシンは処理が正常に終了するまで待機しなくてはなりません。マルチユーザ・マルチプロセス環境では、ロックされたレコードは 上書きされません。

例題 1

以下の例は、asLastNamesとasCompaniesの2つの配列のデータを[People]テーブルにコピーします。配列asLastNamesのデータは[People]Last Nameフィールドに、配列asCompaniesのデータは[People]Companyフィールドに、それぞれ書き込まれます:

```
ARRAY TO SELECTION(asLastNames;[People]Last Name;asCompanies;[People]Company)
```

例題 2

この例題ではレコードセレクションを複製します:

```
ARRAY TEXT(a1;0)
ARRAY TEXT(a2;0)
ARRAY TEXT(a3;0)
ARRAY TEXT(a4;0)

ALL RECORDS([Table_1])
For($i;1;Get last field number(1))
   $p:=Get pointer("a"+String($i))
   SELECTION TO ARRAY(Field(1;$i)->;$p->;*)
 // * を使用してコマンドの実行をスタックする
End for
SELECTION TO ARRAY // この呼び出しで実行が行われる

REDUCE SELECTION([Table_1];0)
For($i;1;Get last field number(1))
   $p:=Get pointer("a"+String($i))
   ARRAY TO SELECTION($p->;Field(1;$i)->;*)
 // * を使用してコマンドの実行をスタックする
End for
ARRAY TO SELECTION // この呼び出しで実行が行われる
```

BOOLEAN ARRAY FROM SET

BOOLEAN ARRAY FROM SET (booleanArr {; set})

引数	型	説明
booleanArr	ブール配列	⇐ レコードがセットに含まれているかいないかを示す配列
set	文字	⇒ セット名、または 引数が省略された場合UserSet

説明

BOOLEAN ARRAY FROM SETコマンドは、テーブル内の各レコードが指定されたセットに含まれるか含まれないかを示すブール配列を作成します。

配列の各要素は、テーブルに作成されたレコードと同じ順序（絶対レコード番号順）で整列されます。配列の0番目の要素はレコード番号0のレコードにあたり、配列の要素NはレコードNのレコードにあたります。

配列の各要素は以下の通りです。

- 対応するレコードがセットに含まれる場合はTrue
- 対応するレコードがセットに含まれない場合はFalse

警告： booleanArr配列の総要素数に意味はありません。構造上の理由により、この数はテーブル上の実存のレコード数と異なっています。起こりうる余分の要素はFalseにセットされます。

set引数を指定しない場合、コマンドはカレントプロセスのUserSetを使用します。

COPY ARRAY

COPY ARRAY (source ; destination)

引数	型	説明
source	配列	⇒コピー元の配列
destination	配列	⇐コピー先の配列

説明

COPY ARRAYコマンドはdestination配列を、source配列と同じ内容、サイズ、およびタイプで作成または上書きします。

sourceとdestinationの配列はローカル、プロセス、インタープロセス配列の組み合わせが可能です。配列の複製時に、これらの変数のスコープの違いによる問題はありません。

例題

以下の例は配列Cを作成し、配列Cと同じ大きさと内容を持つ配列Dを作成します:

```
ALL RECORDS([People]) ` Peopleの全レコードを選択
SELECTION TO ARRAY([People]Company;C) ` companyフィールドのデータをCにコピー
COPY ARRAY(C;D) ` C配列をD配列にコピー
```

Count in array

Count in array (array ; value) -> 戻り値

引数	型		説明
array	配列	⇒	カウントを行う配列
value	式	⇒	カウントする値
戻り値	倍長整数	↩	見つかったインスタンスの数

説明

Count in arrayコマンドは、array内で見つかったvalueの数を返します。

このコマンドは、テキスト、文字、数値、日付、ポインタ、ブールタイプの配列に対して使用できます。引数arrayとvalueは同じタイプか、または互換性があるタイプでなくてはなりません。

valueと一致する項目がarray内に存在しない場合、コマンドは0を返します。

例題

次の例題は、リストボックス内で選択された行の数を表示します:

```
  `tBList はリストボックスの配列名
 ALERT(String(Count in array(tBList;True))+" 行がリストボックスで選択されています。")
```

DELETE FROM ARRAY

DELETE FROM ARRAY (array ; where {; howMany})

引数	型		説明
array	配列	⇒	要素を削除する配列
where	倍長整数	⇒	削除を開始する要素番号
howMany	倍長整数	⇒	削除する要素数, または 省略時は1要素

説明

DELETE FROM ARRAYコマンドは1つまたは複数の要素をarrayから削除します。where引数の示す位置から要素の削除を開始します。

howMany引数は削除する要素数です。howManyが指定されない場合、1つの要素が削除されます。配列のサイズはhowManyだけ小さくなります。

例題 1

以下の例は、配列の5番目の要素から3つの要素を削除します:

```
DELETE FROM ARRAY(anArray;5;3)
```

例題 2

以下の例は、配列の最後の要素を削除します:

```
$vlElem:=Size of array(anArray)
If($vlElem>0)
   DELETE FROM ARRAY(anArray;$vlElem)
End if
```

DISTINCT VALUES

DISTINCT VALUES (aField ; array)

引数	型		説明
aField	フィールド	→	データとして使用する、インデックス可能なフィールド
array	配列	←	フィールドデータを受け取る配列

説明

DISTINCT VALUESコマンドは、aFieldが属するテーブルのカレントセレクションのaFieldフィールドからの重複しない（ユニークな）値で構成されるarray配列を作成します。

DISTINCT VALUESコマンドには、**インデックスが可能な** (インデックスをサポートするタイプの) フィールドを渡すことができます。実際にインデックスされている必要はありません。
しかし、インデックス付けされていないフィールドでこのコマンドを実行すると遅くなります。また、この場合、コマンドはカレントレコードを失うことに注意してください。

DISTINCT VALUESはカレントセレクションのレコードをブラウズし、重複しない値を保持します。

注: トランザクション中に**DISTINCT VALUES**コマンドが呼び出された場合、トランザクション中に作成されたレコードも処理の対象となります。

DISTINCT VALUESを呼び出す前に配列を作成した場合、このコマンドは、その配列タイプがフィールドタイプと互換であることを期待します。そうでない場合、インタプリタモードでは、**DISTINCT VALUES**は適切なタイプの配列を作成します。フィールドが時間型ならば、このコマンドは倍長整数の配列を期待または作成します。

DISTINCT VALUES で使用される配列は、第一引数渡すフィールドと型が一致している必要があります。一致していない場合、配列の型は修正されます。このルールには一つの例外があります: フィールドがピクチャ型である(かつキーワードインデックスと関連付けられている)場合、対応する配列はテキスト型でなければなりません。

コマンドの呼び出し後、配列のサイズはセレクション中の重複しない値の数と同じです。コマンドはカレントセレクションまたはカレントレコードを変更することはありません。**DISTINCT VALUES** コマンドはフィールドのインデックスを使用するので、array中の要素は昇順でソートされて返されます。これが目的の並べ替え順であれば、**DISTINCT VALUES**を使用した後にSORT ARRAYコマンドを呼び出す必要はありません。

注: **DISTINCT VALUES**を キーワードインデックスが適用されたテキストフィールドに対して実行すると、コマンドはインデックスのキーワードで構成される配列を作成します。他のタイ プのデータと異なり、返される値はインデックスの存在により異なります。テキストフィールドの場合、フィールドに標準のインデックスが定義されていても、常にキーワードインデックスが採用されます。テキストやピクチャーフィールドにキーワードインデックスが割り当てられていない場合、空の配列が返されま す。

警告: **DISTINCT VALUES**コマンドは、セレクション及びそこ に含まれる重複しない値の数によって非常に大きな配列を作成する場合があります。配列はメモリ上に存在します。そのためコマンドの実行後、結果をテストす るのは良いことです。これを行うには、作成された配列のサイズをテストするか、ON ERR CALLプロジェクトメソッドを使用してコマンドの呼び出しをカバーします。

4D Server: このコマンドは4D Server用に最適化されています。サーバ側で配列の作成と値の計算が行われ、その後全体がクライアントに送られます。

例題 1

以下の例は、カレントセレクションから都市のリストを作成します。そして、会社の店舗がある都市の数を求めます:

```
ALL RECORDS([Retail Outlets]) ` レコードのセレクションを作成
DISTINCT VALUES([Retail Outlets]City;asCities)
ALERT("会社は"+String(Size of array(asCities))+"都市に店舗を持ちます。")
```

例題 2

"Pictures"フィールドに割り当てられたキーワードインデックスの完全なリストを取得します:

```
ALL RECORDS([PICTURES])
ARRAY TEXT(<>_MyKeywords;10)
DISTINCT VALUES([PICTURES]Photos;<>_MyKeywords)
```

Find in array

Find in array (array ; value {; start}) -> 戻り値

引数	型		説明
array	配列	→	検索を行う配列
value	式	→	配列タイプと同じタイプの検索値
start	倍長整数	→	検索を開始する配列要素番号
戻り値	倍長整数	↩	検索値が見つかった最初の要素番号

説明

Find in array コマンドは、arrayから引数valueと同じものを検索し、最初に発見された要素の番号を返します。

Find in arrayコマンドは、タイプがテキスト、文字列、数値、日付、ポインタ、ブールの配列に使用できます。arrayタイプとvalueのタイプは、必ず同じにしてください。

同じ値を発見できない場合、Find in arrayコマンドは-1を返します。

startを指定すると、その番号の要素から検索を始めます。start引数を指定しない場合、コマンドは第1要素から検索を開始します。

例題 1

以下のプロジェクトメソッドは、文字列またはテキスト配列のポインタを引数として受け、配列から空の要素をすべて削除します:

```
  ` CLEAN UP ARRAY プロジェクトメソッド
  ` CLEAN UP ARRAY (ポインタ)
  ` CLEAN UP ARRAY (->テキストまたは文字配列)

 C_POINTER($1)
 REPEAT
    $vlElem:=Find in array($1->;"")
    If($vlElem>0)
       DELETE FROM ARRAY($1->;$vlElem)
    End if
 Until($vlElem<0)
```

このプロジェクトメソッド実装後、以下のように記述できます:

```
 ARRAY TEXT(atSomeValues;...)
  ` ...
  ` 配列を使用した処理を行う
  ` ...
  ` 空文字の要素を削除する
 CLEAN UP ARRAY(->atSomeValues)
```

例題 2

次のプロジェクトメソッドは、最初の引数で渡した配列(ポインタ指定)中、第二引数で渡した変数やフィールド(ポインタ指定)の値に一致する最初の要素を選択します:

```
  ` SELECT ELEMENT プロジェクトメソッド
  ` SELECT ELEMENT (ポインタ; ポインタ)
  ` SELECT ELEMENT ( ->テキストまたは文字配列; -> テキストまたは文字変数またはフィールド)

 $1->:=Find in array($1->;$2->)
 If($1->=-1)
    $1->:=0  ` 値が要素中に見つからない場合、要素を選択しない
 End if
```

このプロジェクトメソッド実装後、以下のように記述できます:

```
  ` asGender ポップアップメニュオブジェクトメソッド
 Case of
    :(Form event=On Load)
       SELECT ELEMENT(->asGender;->[People]Gender)

 End case
```

INSERT IN ARRAY

INSERT IN ARRAY (array ; where {; howMany})

引数	型		説明
array	配列	→	配列名
where	倍長整数	→	要素を挿入する位置
howMany	倍長整数	→	挿入する要素数, または 省略時は1

説明

INSERT IN ARRAYコマンドは、arrayに1つまたは複数の要素を挿入します。引数whereで指定した要素の前に新しい要素を挿入します。新しく挿入された要素には、配列タイプに応じた空の値が代入されます。whereより後ろの要素は、howManyで指定した数だけ後ろに移動します。

whereが配列サイズより大きい場合、要素は配列の最後に追加されます。

引数howManyは挿入する要素の数を指定します。howManyを省略すると、1つの要素が挿入されます。配列サイズは、howManyだけ大きくなります。

例題 1

以下の例は、配列の10番目の要素位置に新しい5つの要素を挿入します:

```
INSERT IN ARRAY(anArray;10;5)
```

例題 2

以下の例は、配列に要素を追加します:

```
$vlElem:=Size of array(anArray)+1
INSERT IN ARRAY(anArray;$vlElem)
anArray{$vlElem}:=...
```

LIST TO ARRAY

LIST TO ARRAY (list ; array {; itemRefs})

引数	型		説明
list	文字, ListRef	→	一番目の項目をコピーするコピー元のリスト
array	配列	←	コピー先の配列
itemRefs	配列	←	リスト項目の参照番号

説明

LIST TO ARRAYコマンドは、list で指定したリストまたは選択リストの第一レベルの項目で、配列arrayを作成または上書きします。

list 引数には、選択リストの名前(文字列)、または階層リストへの参照(ListRef)を渡すことが出来ます。

コピー先の配列を文字またはテキスト型として事前に設定をしていなかった場合、LIST TO ARRAY は自動的に新しいテキスト配列をデフォルトで作成します。

オプションのitemRefs引数(数値配列)には、リスト項目の参照番号が返されます。

引き続き LIST TO ARRAYコマンドを使い、階層リストの第一レベル要素に基づく配列を構築できます。しかしこのコマンドは子項目を返しません。階層リストで作業する場合、階層リストコマンド、特に Load list の使用が推奨されます。

例題 1

以下の例は、リストRegionsの項目を配列atRegionsにコピーします:

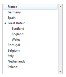
```
LIST TO ARRAY("Regions";atRegions)
```

例題 2

以下のように定義された階層リストについて考えます:

```
myList2:=New list
APPEND TO LIST(myList2;"Scotland";1)
APPEND TO LIST(myList2;"England";2)
APPEND TO LIST(myList2;"Wales";3)
myList1:=New list
APPEND TO LIST(myList1;"France";1)
APPEND TO LIST(myList1;"Germany";2)
APPEND TO LIST(myList1;"Spain";3)
APPEND TO LIST(myList1;"Great Britain";4;myList2;True)
APPEND TO LIST(myList1;"Portugal";5)
APPEND TO LIST(myList1;"Belgium";6)
APPEND TO LIST(myList1;"Italy";7)
APPEND TO LIST(myList1;"Netherlands";8)
APPEND TO LIST(myList1;"Ireland";9)
```

このリストは以下の様に表示されます:



```
France
Germany
Spain
Great Britain
    Scotland
    England
    Wales
Portugal
Belgium
Italy
Netherlands
Ireland
```

これに対し以下の宣言を実行すると、:

```
LIST TO ARRAY(myList1;$MyArray)
```

……結果は以下の様になります。

```
$MyArray{1}="France"
$MyArray{2}="Germany"
$MyArray{3}="Spain"
$MyArray{4}="Great Britain"
$MyArray{5}="Portugal"
...
```

LONGINT ARRAY FROM SELECTION

LONGINT ARRAY FROM SELECTION (aTable ; recordArray {; selection})

引数	型		説明
aTable	テーブル	→	カレントセレクションのテーブル
recordArray	倍長整数配列	←	レコード番号配列
selection	文字	→	命名セレクション名、または 省略した場合カレントセレクション

説明

LONGINT ARRAY FROM SELECTION コマンドは、selectionの（絶対）レコード番号をrecordArrayに返します。

selection引数を省略した場合、コマンドはtableのカレントセレクションを使用します。

Note: 配列要素0は-1に初期化されます。

MULTI SORT ARRAY

MULTI SORT ARRAY (array {; sort}{; array2 ; sort2 ; ... ; arrayN ; sortN})
引数 型 説明
array 配列 ⇨ソートする配列
sort 演算子⇨">" : 昇順ソート, または "<" : 降順ソート, または 省略した場合、ソートしない
MULTI SORT ARRAY (ptrArrayName ; sortArrayName)
引数 型 説明
ptrArrayName ポインター配列 ⇨配列ポインタの配列
sortArrayName 倍長整数配列⇨ソート順配列 (1 = 昇順にソート、-1 = 降順にソート、0 = 前のソートに同期)

説明

MULTI SORT ARRAY コマンドにより、一連の配列に対してマルチレベルソートを実行することができます。

このコマンドは2種類の構文を受け入れます。

- **第一の構文 : MULTI SORT ARRAY (array{; sort}{; array2; sort2; ...; arrayN; sortN})**

この構文が最もシンプルです。複合条件による並び替えを適用しようとする同期した配列の名前を直接渡すことができます。

配列だけを渡すか、あるいは (配列;> または <) の組み合わせを必要なだけ渡すことができます。引数として渡された配列はすべて、同期化されて並び替えられます。

ポインタタイプやピクチャタイプの配列を除き、任意のタイプの配列を渡すことができます。二次元配列の要素（例えばa2DArray{$vlThisElement}）を並び替えられますが、二次元配列自体を並び替えることはできません（例えばa2DArray）。

配列の内容を並び替え条件として使用するには、sort 引数を渡します。この引数の値 (> または <) により、その配列の並び替え順序（昇順または降順）が決まります。sort引数を省略した場合、その配列の内容は並び替え条件として使用されません。

Note: コマンドが正常に機能するには、少なくとも1つの並び替え条件を渡さなければならないということを覚えておいてください。並び替え条件が設定されていない場合はエラーが生成されます。

コマンドに渡した配列の順番により、並び替えレベルが決定します。つまりシンタックス中、並び替え条件が指定された配列の位置により、その並び替えレベルが決まります。

- **・第二の構文 : MULTI SORT ARRAY (ptrArrayName; sortArrayName)**

この構文はより複雑であり、汎用的な開発では非常に有効です（例えば、あらゆるタイプの配列を並び替える汎用メソッドを作成したり、汎用的なSORT ARRAYコマンドに相当するものを作成する場合）。

ptrArrayName引数には、配列ポインタの配列名を指定します。この配列の各要素は、並び替える配列を示すポインタです。ptrArrayNameに指定した配列ポインタの順に、並び替えが実行されます。

警告 : ptrArrayNameがさすすべての配列は同じ要素数でなければなりません。

Note: ptrArrayName は、ローカル（$ptrArrayName）、プロセス（ptrArrayName）、インタープロセス（<>ptrArrayName）タイプのポインタの配列を指定することができます。これとは逆に、この配列の要素が指す対象は、プロセス配列またはインタープロセス配列でなくてはなりません。

sortArrayName引数には配列名を渡し、この配列の各要素は対応するポインタ配列要素の並び替え順(-1、0または1)を示します:
・-1=降順並び替え
・0=配列は並び替え条件として使用されませんが、他の並び替えに応じて並び替えられます。
・1=昇順並び替え

Note: ポインタタイプやピクチャタイプの配列を並び替えることはできません。二次元配列（例えばa2DArray{$vlThisElement}）の要素を並び替えることができますが、二次元配列そのもの（例えばa2DArray）を並び替えることはできません。

ptrArrayName配列の各要素に対して、対応するsortArrayName配列の要素が存在していなければなりません。したがって、必ずこの2つの配列の要素数は同じになります。

例題 1

次の例題では1番目の構文を使用します。4つの配列を作成して都市（昇順）そして給与（降順）で並び替えます。最後の2つの配列、namesとtelNumsは、前の並び替え条件に応じて同期されます:

```
ALL RECORDS([Employees])
SELECTION TO ARRAY([Employees]City;cities;[Employees]Salary;salaries;[Employees]Name;
names;[Employees]TelNum;telNums)
MULTI SORT ARRAY(cities;>;salaries;<;names;telNums)
```

配列namesを3番目の並び替え条件として使用したい場合は、引数であるnames配列の後に > または < を付加します。

このシンタックスは:

```
MULTI SORT ARRAY(cities;>;salaries;names;telNums)
```

以下と同じです:

```
SORT ARRAY(cities;salaries;names;telNums;>)
```

例題 2

次の例題では2番目の構文を使用します。4つの配列を作成し、都市（昇順）と会社（降順）で並び替えます。最後の2つの配列、names配列とtelNums配列は、前の並び替え条件に応じて同期化されます:

```
ALL RECORDS([Employees])
SELECTION TO ARRAY([Employees]City;cities;[Employees]Company;companies;[Employees]Name;
names;[Employees]TelNum;telNums)
ARRAY POINTER(pointers_Array;4)
ARRAY LONGINT(sorts_Array;4)
pointers_Array{1}:=->cities
sorts_Array{1}:=1
pointers_Array{2}:=->companies
sorts_Array{2}:=-1
pointers_Array{3}:=->names
sorts_Array{3}:=0
pointers_Array{4}:=->telNums
sorts_Array{4}:=0
MULTI SORT ARRAY(pointers_Array;sorts_Array)
```

3番目の並び替え条件としてnames配列を使用したい場合には、sorts_Array {3}要素に値1を割り当てる必要があります。または、都市だけを条件として配列を並び替えたい場合は、sorts_Array {2}、sorts_Array {3}、sorts_Array {4}の要素に値0を割り当てます。すると、SORT ARRAY(cities;companies;names;telNums;>)と同じ結果を得ることができます。

SELECTION RANGE TO ARRAY

SELECTION RANGE TO ARRAY (start ; end {; field | table ; array} {; field | table2 ; array2 ; ... ; field | tableN ; arrayN})

引数	型		説明
start	倍長整数	⇨	データ取得を開始するレコード位置番号
end	倍長整数	⇨	データ取得を終了するレコード位置番号
field | table	フィールド, テーブル	⇨	データを取得するフィールドまたは レコード番号を取得するテーブル
array	配列	⇦	フィールド値またはレコード番号を受け取る配列

説明

SELECTION RANGE TO ARRAYコマンドは1つまたは複数の配列を作成し、その配列にカレントセレクションのフィールド値またはレコード番号を代入します。

カレントセレクション全体を対象とするSELECTION TO ARRAYと異なり、**SELECTION RANGE TO ARRAY**コマンドは、セレクション中、引数startとendによって指定されたレコード位置番号の範囲が適用範囲となります。

このコマンドは、startとendが、1 <= start <= end <= Records in selection ([...])の式の条件を満たしていることを期待します。

1 <= start = end < Records in selection ([...])を渡すと、start = endで選択されたレコードのフィールドをロード、あるいはレコード番号を取得します。

間違ったレコード位置番号を渡すと、このコマンドは以下を実行します:

- end > Records in selection ([...])の場合、startで指定されたレコードから最後のレコードまでの値を返します。
- start > endの場合、startで指定されたレコードのみの値を返します。
- 両方の引数がセレクションのサイズと一致しない場合、空の配列を返します。

SELECTION TO ARRAYコマンド同様、**SELECTION RANGE TO ARRAY**コマンドも最初に指定したテーブルのセレクションを用います。

またSELECTION TO ARRAYコマンド同様、**SELECTION RANGE TO ARRAY**コマンドは以下の動作を行います:

- 1つまたは複数のフィールドから値をロードします。
- シンタックス...;[テーブル];配列;...を使用して、レコード番号をロードします。
- テーブル間にn対1の自動リレートが設定されている場合や、SET AUTOMATIC RELATIONSを事前に呼び出してn対1のマニュアルリレートを自動に変更した場合は、リレートフィールドの値をロードします。いずれの場合も、複数レベルのn対1リレートを経てテーブルから値がロードされます。

各配列は、そのフィールドタイプに応じてタイプ定義されます。

SELECTION RANGE TO ARRAY を時間型のフィールドに対して適用した場合、配列が他の型として定義されたことがない場合のみ時間型配列が作成されるという点に注意して下さい。例えば、以下のコンテキストでは、*myArray* 配列はその後も倍長整数型配列のままです:

```
ARRAY LONGINT(myArray;0)
SELECTION TO ARRAY([myTable]myTimeField;myArray)
```

レコード番号をロードする場合、配列のタイプは倍長整数となります。

注: start と end 引数のみを指定して**SELECTION RANGE TO ARRAY**コマンドを呼び出すことができます。この特別なシンタックスを使用して、* を指定した一連のSELECTION TO ARRAYコマンドスタックを、限定してセレクションに対し実行することができます (例題4参照)。

4D Server: SELECTION RANGE TO ARRAYコマンドは4D Server用に最適化されています。各配列はサーバ上で作成され、配列全体がクライアントマシンに送信されます。

警告: SELECTION RANGE TO ARRAYコマンドは、startとendで 指定した範囲やロードするデータサイズによって非常に大きな配列を作成する場合があります。配列はメモリ上に存在します。そのためコマンドの実行後、結果 をテストするのは良いことです。これを行うには、作成された配列のサイズをテストするか、ON ERR CALLを使用してコマンドの呼び出しをカバーします。

コマンドが正常に実行されると、結果配列のサイズは(end-start)+1になります。ただしend引数がセレクションのレコード数より大きい場合、結果の配列は(Records in selection([...])-start)+1個の要素を含みます。

例題 1

以下の例は、[Invoices]テーブルのカレントセレクションの先頭から50レコードを選択します。そして、[Invoices]Invoice IDフィールドおよびリレートフィールドの[Customers]Customer IDから値をロードします:

```
SELECTION RANGE TO ARRAY(1;50;[Invoices]Invoice ID;alInvoID;[Customers]Customer ID;alCustID)
```

例題 2

以下の例は、[Invoices]テーブルのカレントセレクションの最終50レコードを選択します。そして、[Invoices]レコードおよび[Customers]リレートレコードのレコード番号をロードします:

```
lSelSize:=Records in selection([Invoices])
SELECTION RANGE TO ARRAY(lSelSize-49;lSelSize;[Invoices];alInvRecN;[Customers];alCustRecN)
```

例題 3

以下の例は、配列に一度に全体をダウンロードできないかもしれない、大きなセレクションのレコードを、1000づつシーケンシャルに処理します:

```
lMaxPage:=1000
lSelSize:=Records in selection([Phone Directory])
For($lPage ;1;1+((lSelSize-1)\lMaxPage))
  ` 値やレコード番号をロード
   SELECTION RANGE TO ARRAY(1+(lMaxPage*($lPage-1));lMaxPage*$lPage;...;...;...;...;...;...)
  ` 配列に対し処理を行う
End for
```

例題 4

[Invoices]テーブルのカレントセレクション中1~50番目のレコードのみを配列に取得します:

```
  // 取得するフィールドと配列のペアを指定
SELECTION TO ARRAY([Invoices]InvoiceRef;arrLInvRef;*)
SELECTION TO ARRAY([Invoices]Date;arrDInvDate;*)
SELECTION TO ARRAY([Clients]ClientRef;arrLClientRef;*)
  // データの取得を実行
SELECTION RANGE TO ARRAY(1;50)
```

SELECTION TO ARRAY

SELECTION TO ARRAY {(field | table ; array {; aField ; array {; aField2 ; array2 ; ... ; aFieldN ; arrayN}}{; *})}

引数	型		説明
field | table	フィールド, テーブル	→	データを取得するフィールドまたは レコード番号を取得するテーブル
array	配列	←	フィールド値またはレコード番号を受け取る配列
aField	フィールド	→	配列に値を取得するフィールド
array	配列	←	フィールドデータを受け取る配列
*	演算子	→	実行をスタックする

説明

SELECTION TO ARRAYコマンドは、1つ以上の配列を作成し、カレントセレクションのフィールドデータやレコード番号を配列にコピーします。

SELECTION TO ARRAYコマンドは第一引数 (テーブル参照またはフィールド参照) で指定されたテーブルのセレクションに適用されます。**SELECTION TO ARRAY**は以下のことを行います:

- 1つまたは複数のフィールドから値をロードします。
- [table];array シンタックスを使用してテーブルからレコード番号をロードします。
- テー ブル間にn対1の自動リレートが設定されている場合や、SET AUTOMATIC RELATIONSを事前に呼び出してn対1のマニュアルリレートを自動に変更した場合は、リレートフィールドの値をロードします。いずれの場合も、複 数レベルのn対1リレートを経てテーブルから値がロードされます。

各配列は、そのフィールドタイプに応じてタイプ定義されます。

SELECTION TO ARRAY を時間型のフィールドに対して適用した場合、配列が他の型として定義されたことがない場合のみ時間型配列が作成されるという点に注意して下さい。例えば、以下のコンテキストでは、*myArray* 配列に対して適用した場合、配列はその後も倍長整数型配列のままです:

```
ARRAY LONGINT(myArray;0)
SELECTION TO ARRAY([myTable]myTimeField;myArray)
```

レコード番号をロードする場合、配列のタイプは倍長整数となります。

* 引数を渡すと、4Dはその行の実行を遅延し、メモリに格納します。* で終わる行を使用して複数の行をスタックできます。スタックされた行は * なしの**SELECTION TO ARRAY**一回の呼び出しですべて実行されます。この目的のため、このコマンドを引数なしで呼び出すことができます。
これにより、QUERYコマンドのように、複雑な文を複数の行に分割して記述することができ、可読性が向上します。また途中の行の挿入も容易です(ARRAY TO SELECTION コマンドの例題2参照)。

4D Server: SELECTION TO ARRAYコマンドは4D Server用に最適化されています。各配列はサーバ上で作成され、配列全体がクライアントマシンに送信されます。

警告 : SELECTION TO ARRAYコマンドは、カレントセレクションの大きさやロードするデータサイズによって非常に大きな配列を作成する場合があります。配列はメモリ上に存在します。そのためコマンドの実行後、結果をテストするのは良いことです。これを行うには、作成された配列のサイズをテストするか、ON ERR CALLを使用してコマンドの呼び出しをカバーします。

注: SELECTION TO ARRAYコマンドを呼び出した後、カレントセレクションとカレントレコードは同じままですが、カレントレコードはロードされていません。もしカレントレコードのフィールド値を使いたい場合は**SELECTION TO ARRAY**コマンドの後にLOAD RECORDコマンドを使用してください。

例題 1

以下の例は、[People]テーブルと[Company]テーブルは自動リレーションを持ちます。2つの配列asLastNameとasCompanyAddrは、[People] テーブルのセレクションの数にリサイズされ、両テーブルからのデータを受け取ります:

```
SELECTION TO ARRAY([People]Last Name;asLastName;[Company]Address;asCompanyAddr)
```

例題 2

以下の例は、[Clients]レコード番号を配列alRecordNumbersに、[Clients]Names フィールドの値を配列asNames代入します:

```
SELECTION TO ARRAY([Clients];alRecordNumbers;[Clients]Names;asNames)
```

Size of array

Size of array (array) -> 戻り値

引数	型		説明
array	配列	→	サイズを取得する配列
戻り値	倍長整数	↩	配列の要素数

説明

Size of array コマンドは、配列arrayの要素数を返します。

例題 1

以下の例は配列anArrayのサイズを返します:

```
vlSize:=Size of array(anArray) ` vlSizeはanArrayのサイズを取得する
```

例題 2

以下の例は2次元配列の行数を返します:

```
vlRows:=Size of array(a2DArray) ` vlRowsはa2DArrayのサイズを取得する
```

例題 3

以下の例は2次元配列の、指定された行の列数を返します:

```
vlColumns:=Size of array(a2DArray{10}) ` vlColumnsはa2DArray{10}のサイズを所得する
```

SORT ARRAY

SORT ARRAY (array {; array2 ; ... ; arrayN}{; > または <})

引数	型	説明
array	配列	→ ソートする配列
> または <	演算子	→ ">" : 昇順ソート, または "<" : 降順ソート, または 省略した場合降順ソート

説明

SORT ARRAYコマンドは、1つまたは複数の配列を昇順や降順にソートします。

注: Pointer 配列や Picture 配列のソートは行えません。二次元配列の要素(例えば a2DArray{$vlThisElem})をソートすることはできますが、二次元配列そのもの(a2DArray)をソートすることはできません。

最後の引数は、ソートの昇順または降順を指定します。引数に大なり記号(>)を指定すると昇順にソートします。引数に小なり記号(<)を指定すると降順にソートします。省略した場合は昇順にソートします。

複数の配列を指定した場合、すべての配列が最初の配列の順番でソートされます。各配列ごとに個々にソートするわけではありません。その代り、同期した配列をソートしたい場合には MULTI SORT ARRAY コマンドを使用することができます。

例題 1

以下の例は2つの配列を作成し、companyでソートします:

```
ALL RECORDS([People])
SELECTION TO ARRAY([People]Name;asNames;[People]Company;asCompanies)
SORT ARRAY(asCompanies;asNames;>)
```

しかし、SORT ARRAY コマンドはマルチレベルソートを行わないため、会社ごとの従業員名の順序はばらばらのままです。各会社毎に従業員名をソートするには、次のようにします:

```
ALL RECORDS([People])
ORDER BY([People];[People]Company;>;[People]Name;>)
SELECTION TO ARRAY([People]Name;asNames;[People]Company;asCompanies)
```

例題 2

次の例では、[People]テーブルから名前をフローティングウィンドウに表示します。ウィンドウ上のボタンをクリックすると、名前のリストをAからZへ、またはZからAへソートします。同じ名前の人が複数存在するため、インデックス付きで重複不可である[People]ID numberフィールドを使用できます。名前のリストをクリックすると、クリックした名前を持つレコードを取得します。同期がとられ、表示されないalIDs配列を利用することにより、必ずクリックした名前に対応するレコードにアクセスできます:

```
  ` asNames配列オブジェクトメソッド
Case of
   :(Form event=On Load)
      ALL RECORDS([People])
      SELECTION TO ARRAY([People]Name;asNames;[People]ID number;alIDs)
      SORT ARRAY(asNames;alIDs;>)
   :(Form event=On Unload)
      CLEAR VARIABLE(asNames)
      CLEAR VARIABLE(alIDs)
   :(Form event=On Clicked)
      If(asNames#0)
  ` 正しいレコードを取得するためにalIDsを使用する
         QUERY([People];[People]ID Number=alIDs{asNames})
  ` レコードの処理を行う
      End if
End case

  ` bA2Z ボタンオブジェクトメソッド
  ` 配列を昇順でソートし、かつ同期を保つ
SORT ARRAY(asNames;alIDs;>)

  ` bZ2A ボタンオブジェクトメソッド
  ` 配列を降順でソートし、かつ同期を保つ
SORT ARRAY(asNames;alIDs;<)
```

TEXT TO ARRAY

TEXT TO ARRAY (varText ; arrText ; width ; fontName ; fontSize {; fontStyle {; *}})

引数	型		説明
varText	テキスト	→	分割する元のテキスト
arrText	テキスト配列	←	単語または行に分割されたテキストを受け取る配列
width	倍長整数	→	文字列の最大幅 (ピクセル)
fontName	テキスト	→	フォント名
fontSize	倍長整数	→	フォントサイズ
fontStyle	倍長整数	→	フォントスタイル
*	演算子	→	指定時、テキストをマルチスタイルとして解釈する

説明

TEXT TO ARRAYコマンドはテキスト変数をテキスト配列変換します。元のvarTextテキストは (スタイル付きであってもなくても) 分割され、arrText配列の要素となり、コマンドから返されます。このコマンドを使用して例えば適切な位置でテキストが分割されたメール本文を作成することができます。

テキストの分割はコマンドに渡される行幅やテキスト属性に基づいて計算されます。

varText 引数には配列要素に分解する元のテキストを渡します。このテキストはマルチスタイルであってもなくてもかまいません。マルチスタイルの場合、いくつかの引数は無視されます。

arrText 引数にはテキスト配列を渡します。この配列には分割されたテキストを要素とするテキスト配列が返されます。

width 引数にはテキストを分割する際のサイズ計算に用いる行の幅をピクセル単位で返します。テキスト全体に対しテキストの描画属性 (フォント名やサイズ、スタイル) を考慮に入れたうえで、コマンドはこのサイズに収まる単語の切れ目を計算します。

- マルチスタイルテキストの場合、元のテキストのスタイルが計算に使用され、以降の引数はコマンドに渡されても無視されます。この場合結果配列に格納されるテキストは元のスタイルを保持します。
- スタイルなしのテキストの場合、コマンドが行の長さを決定できるようにするため、すべての引数を渡さなければなりません。

各配列要素には最低一つの単語が含まれます。width に渡された値が分割ルールに対し小さすぎる場合、配列は渡された引数に基づいて可能な限り計算されて作成され、OK変数に0が返されます。例えば3 ピクセル幅を渡した場合、ほとんどの単語はこの長さより大きくなるでしょう。このようなケースではOK変数が0となります。
4Dが単語の切れ目を認識できないような場合には、最大幅を超えた要素が作成されることもある点に留意してください。
またこれは配列の最大際要素数は論理的にvarTextの単語数になることを意味します。

fontName と fontSize 引数にはそれぞれ、varText の分割計算の際に使用するフォント名とサイズを渡します。これらの引数はスタイルなしテキストの場合に必須です。

fontStyle 引数にはFont Stylesテーマの以下の定数を1つ以上渡せます:

定数	型	値
Bold	倍長整数	1
Italic	倍長整数	2
Plain	倍長整数	0
Underline	倍長整数	4

この引数は省略可能です。省略した場合Normalスタイルが使用されます。

オプションの * 引数を指定すると、マルチスタイルテキストの場合で、元のテキストにテキスト属性が指定されていない場合、fontName, fontSize そして fontStyle 引数に渡した値を使用するよう指示することができます。しかしこれらの属性が元のテキスト内で定義されていた場合は無視されます。

例題 1

スタイル付きテキストを200ピクセル幅で分割します:

```
TEXT TO ARRAY(theText;TextArray;200;"Arial";20;Normal;*)
 // Arial, 20, そして Normal 属性はマルチスタイルテキスト中でこれらの属性が指定されていない場合にのみ使用されます
```

例題 2

スタイルなしテキストを350ピクセル幅、Bodoni Boldフォント、サイズ14で分割します。フォントが利用できない場合コマンドは正しく動作しないので、フォントの存在をチェックしなければなりません:

```
ARRAY TEXT($FontList;0)
FONT LIST($FontList)
$Font:="Bodoni"
$p:=Find in array($FontList;$Font)
If($p>0)
   TEXT TO ARRAY(theText;TextArray;350;"Bodoni";14;Bold)
Else
 // 他のフォントを使用する
End if
```

例題 3

スタイル付きテキストをスタイルなしのArial Normalフォント、サイズ12、最大幅600ピクセルで印刷したいとします:

```
 // マルチスタイルテキストをスタイルなしテキストに変換します:
$RawText:=OBJECT Get plain text(vText)
 // 配列を作成します
TEXT TO ARRAY($RawText;TextArray;600;"Arial";12)
```

例題 4

400ピクセル幅のエリアを印刷します。テキストは最大80行で可能な限り大きなフォント (最大24ポイント) を使用します:

```
ARRAY TEXT(TextArray;0)
$Size:=24
Repeat
   TEXT TO ARRAY($RawText;TextArray;400;"Arial";$Size)
   $Size:=$Size-1
   $n:=Size of array(TextArray)
Until($n<=80)
```

ARRAY STRING

ARRAY STRING (strLen ; arrayName ; size {; size2})

引数	型		説明
strLen	倍長整数	→	文字長 (1... 255)
arrayName	配列	→	配列名
size	倍長整数	→	配列の要素数、またはsize2を指定した場合は配列の行数
size2	倍長整数	→	2次元配列の列数

説明

ARRAY STRINGコマンドは、メモリ上に文字列要素の配列を作成またはリサイズします。

互換性に関する注意: 4Dバージョン11以降で作成されたデータベースはデフォルトでUnicodeモードです (ASCIIコード参照)。このモードではARRAY STRINGコマンドの動作は完全にARRAY TEXTコマンドと同じです (strLen引数は無視される)。新規開発ではARRAY TEXTの使用を推奨します。ARRAY STRINGコマンドは互換性のためにのみ保持されています。

- strLen引数は、文字列配列のそれぞれの要素に格納できる最大文字長です。長さは1から255バイトを指定できます。
  注: この引数はデータベースが非Unicodeモードで動作している場合にのみ考慮されます。Unicodeモードでは無視されます (互換性に関する注意参照)。
- arrayName引数は作成する配列の名前です。
- size引数は配列の要素数です。
- size2引数はオプションです。size2が渡されている場合、コマンドは2次元配列を作成します。この場合、sizeに配列の行数を、size2にはそれぞれの配列の列数を指定します。2次元配列の各行は要素および配列として扱えます。これは配列の一番目の次元を扱う時、2次元配列中の配列全体を挿入あるいは削除するために、他の配列コマンドを使用できることを意味します。

ARRAY STRINGを既存の配列に適用する場合、

- 配列サイズを増やす場合、既存の値は保持され、新しい要素は空の文字列 "" で初期化されます。
- 配列サイズを減らす場合、後ろの要素は配列から削除され、失われます。

例題 1

この例は、31文字100要素の文字列プロセス配列を作成します:

```
ARRAY STRING(31;aSValues;100)
```

例題 2

この例は、63文字100行50列要素の文字列ローカル配列を作成します:

```
ARRAY STRING(63;$aSValues;100;50)
```

例題 3

この例は、255文字50要素の文字列インタープロセス配列を作成し、それぞれの要素に"Element #"および要素番号文字列を格納します:

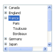
```
ARRAY STRING(255;<>asValues;50)
For($vlElem;1;50)
   <>asValues{$vlElem}:="Element #"+String($vlElem)
End for
```

階層リスト

- 階層リストの管理
- APPEND TO LIST
- CLEAR LIST
- Copy list
- Count list items
- DELETE FROM LIST
- Find in list
- GET LIST ITEM
- Get list item font
- GET LIST ITEM ICON
- GET LIST ITEM PARAMETER
- GET LIST ITEM PARAMETER ARRAYS
- GET LIST ITEM PROPERTIES
- GET LIST PROPERTIES
- INSERT IN LIST
- Is a list
- List item parent
- List item position
- LIST OF CHOICE LISTS
- Load list
- New list
- SAVE LIST
- SELECT LIST ITEMS BY POSITION
- SELECT LIST ITEMS BY REFERENCE
- Selected list items
- SET LIST ITEM
- SET LIST ITEM FONT
- SET LIST ITEM ICON
- SET LIST ITEM PARAMETER
- SET LIST ITEM PROPERTIES
- SET LIST PROPERTIES
- SORT LIST
- REDRAW LIST

階層リストの管理

階層リストはフォームオブジェクトで、拡張/折り畳み可能な複数レベルのデータリスト表示に使用できます。



フォーム中で、階層リストはデータの表示および入力に使用できます。リスト項目ごとに20億文字 (テキストフィールドの最大サイズ) までの文字を含めることができ、アイコンを割り当てることができます。一般的に階層リストはクリック、ダブルクリック、キーボードナビゲーション、そしてドラッグ＆ドロップをサポートします。リストの内容を検索することも可能です (Find in list コマンド)。

作成と更新

階層リストは(New list または Copy list コマンドを使用することによって)プログラミングで全て作成することが出来ます。また、デザインモードのリストエディターで定義されたリストを使用(Load list コマンド)することで作成することもできます。

階層リストの内容やアピアランスは、"階層リスト"テーマのコマンドを使用して、プログラムで管理できます。特定のアピアランス等については オブジェクトプロパティ テーマのコマンドで設定できます (後述)。

OBJECT SET LIST BY REFERENCE または OBJECT SET LIST BY NAME コマンドを使用することによって、階層リスト参照をフォームオブジェクトの選択リスト(ソース、指定リスト、除外リスト)と、動的に関連付けることもできます。また、プロパティリストを使用することによって、リストエディターにて定義された選択リストをフォームオブジェクトと関連付けることもできます。

ListRef とオブジェクト名

階層リストはメモリ上に存在する**ランゲージオブジェクト**であり、同時に**フォームオブジェクト**でもあります。

ランゲージオブジェクトは倍長整数型のユニークな内部IDで参照されます (本マニュアル中ではListRefと表記)。このIDはリストを作成するNew list、Copy list、Load list、BLOB to listから返されます。メモリ中にランゲージオブジェクトのインスタンスはひとつのみしか存在せず、このオブジェクトに対して行われた変更は即座に使用されているすべての場所に適用されます。

フォームオブジェクトはユニークである必要はありません。同じ階層リストを同じフォームや異なるフォーム上で使用できます。他のフォームオブジェクト同様、ランゲージ中でシンタックス (*;"ListName") を使用してオブジェクトを指定します。

ランゲージオブジェクトとしての階層リストと、フォームオブジェクトとしての階層リストは、ListRef の値を格納した中間的な変数により接続されます。例えば:

```
mylist:=New list
```

のように記述するとListRefがmylistに格納され、階層リストフォームオブジェクトのプロパティリストにmylist変数名を記述でき、ランゲージオブジェクトを管理できます。

リストはフォームオブジェクトごとに個別の性質を、および他のリストフォームオブジェクトと共有される性質を持ちます。以下の性質はリストフォームオブジェクトごとに固有のものです:

- 選択された項目
- 項目の拡張/折りたたみ状況
- スクロールカーソルの位置

他の性質 (フォント、フォントサイズ、スタイル、入力コントロール、カラー、リストの内容、アイコン、その他) は他のリストフォームオブジェクトと共有され、個別に変更することはできません。

したがって、拡張/折りたたみ状況に基づくコマンドやカレントの項目に関するコマンド、例えば Count list items (最後の * 引数を渡さずに) を使用するとき、どのフォームオブジェクトに対する処理なのかを明示的に示すことが重要です。

メモリ中の階層リストを指定する場合は、ランゲージコマンドにListRef IDを使用しなければなりません。
フォームレベルで階層リストのオブジェクトを指定する場合は、コマンド中でシンタックス (*;"ListName") を使用して、オブジェクト名を指定します。このシンタックスは"オブジェクトプロパティ"テーマのコマンドで使用されるものと同じで、リストのプロパティに対して動作する"階層リスト"テーマのほとんどのコマンドで受け入れられます (このテーマのコマンドの説明を参照してください)。

警告, プロパティを設定するコマンドの場合、オブジェクト名に基づくシンタックスは、指定されたフォームオブジェクトのみがコマンドで変更されることを意味せず、それよりもコマンドの動作はこのオブジェクトの状態に基づきます。階層リスト共有の性質は常にすべてのオブジェクトで変更されます。

例えば、SET LIST ITEM FONT(*;"mylist1";*;thefont)を実行すると、mylist1フォームオブジェクトに関連付けられた階層リスト項目のフォントを変更します。コマンドはmylist1オブジェクトの現在選択されている項目を更新の対象としますが、変更はすべてのプロセスのすべてのリストに対して実行されます。

@のサポート

他のオブジェクトプロパティ管理コマンドのように、ListName 引数で "@" 文字を使用できます。このシンタックスはフォーム中のオブジェクトグループを指定するために使用されます。しかし階層リストコマンドのコンテキストでは、これはすべてのケースで適用されるわけではありません。このシンタックスはコマンドのタイプにより、2つの異なる効果があります:

- プロパティを設定するコマンドにおいて、このシンタックスは対応する名前すべてのオブジェクトを対象とします (標準の動作)。例えば引数 "LH@" はオブジェクト名が"LH"で始まる階層リストを指定します。以下のコマンドがこの動作となります:
  DELETE FROM LIST
  INSERT IN LIST
  SELECT LIST ITEMS BY POSITION
  SET LIST ITEM
  SET LIST ITEM FONT
  SET LIST ITEM ICON
  SET LIST ITEM PARAMETER
  SET LIST ITEM PROPERTIES
- プロパティを取得するコマンドにおいて、このシンタックスは対応する名前で最初に見つかったオブジェクトを対象とします。以下のコマンドがこの動作となります:
  Count list items
  Find in list
  GET LIST ITEM
  Get list item font
  GET LIST ITEM ICON
  GET LIST ITEM PARAMETER
  GET LIST ITEM PROPERTIES
  List item parent
  List item position
  Selected list items

階層リストに対し利用できる汎用コマンド

4Dの汎用コマンドを使用して、フォーム上の階層リストオブジェクトのアピアランスを変更できます。これらのコマンドには (* 引数を使用して) 階層リストオブジェクト名または (標準シンタックスの) 変数名を渡します。

Note: 階層リストの場合、フォームの変数はListRef値を含んでいます。階層リストに関連付けられた変数を渡して属性を変更するコマンドを実行すると、複数の階層リストフォームオブジェクトがある時に、それらの一つを特定することができません。それぞれ異なる表示をさせたい場合はオブジェクト名で指定します。

以下は階層リストに対して適用可能なコマンドのリストです:
OBJECT SET FONT
OBJECT SET FONT STYLE
OBJECT SET FONT SIZE
OBJECT SET COLOR
OBJECT SET FILTER
OBJECT SET ENTERABLE
OBJECT SET SCROLLBAR VISIBLE
OBJECT SET SCROLL POSITION
OBJECT SET RGB COLORS

Reminder: OBJECT SET SCROLL POSITION コマンドを除き、オブジェクト名を指定したとしても、これらのコマンドは同じリストのすべての表示を変更します。

プロパティコマンドの優先順位

階層リストの特定のプロパティ (例えば入力可属性やカラーなど) は3つの異なる方法で設定できます: デザインモードのプロパティリスト、"オブジェクトプロパティ" テーマのコマンド、"階層リスト" テーマのコマンド。
これら3つの方法すべてをプロパティ設定に使用した場合、以下の優先順位が適用されます:

1. "階層リスト" テーマのコマンド
2. 汎用のオブジェクトプロパティコマンド
3. プロパティリストのパラメタ

この原則は、コマンドが呼び出された順番に関係なく適用されます。階層リストコマンドを使用して個々に項目プロパティを変更すると、同等のオブジェクトプロパティコマンドは、そのあとに呼び出されたとしても、その項目に対しては効果を持たなくなります。例えばSET LIST ITEM PROPERTIES コマンドを使用して項目のカラーを変更すると、この項目に対してOBJECT SET COLOR コマンドは効果を持たなくなります。

位置あるいは参照による項目の管理

階層リストのコンテンツにアクセスするには通常2つの方法、位置または参照のいずれかを使用します。

- 位置を使用する場合、項目を特定する ために、スクリーン上に表示されているリストの項目に関連する位置を基とします。結果は階層項目が拡げられているか折りたためられているかにより異なりま す。複数のフォームオブジェクトがある場合、フォームオブジェクトごとに拡張/折りたたみの状態が異なることに留意してください。
- 参照を使用する場合、リスト項目のitemRef IDを参照します。これにより、それぞれの項目を階層リスト中での位置や表示状態に関わらず特定できます。

項目参照番号を使用する (itemRef)

階層リストのそれぞれの項目は倍長整数型の参照番号 (itemRef) を持ちます。この値は開発者が使用します。4Dはたんに番号を保存するだけです。

警告: どの倍長整数値も参照として使用できますが、0だけは特別な意味を持ちます。このテーマのほとんどのコマンドで、0は最後にリストに追加された項目を指定するために使用されます。

参照番号を使用するにあたりいくつかTipsを紹介します:

1. 項目をユニーク値で識別する必要がない場合 (初心者レベル).
   - 最初の例としてアドレスブックで使用するタブシステムを構築するとします。システムは選択されたタブの番号を返すので、それ以上の情報は必要ありません。この場合、項目参照番号について心配する必要はありません。0以外の値をitemRefに渡します。アドレスブックシステムの場合、デザインモードでA-Zのリストを定義することができる点に留意してください。またプログラムを使用して、レコードがない文字を取り除いたリストを作成することもできます。
   - キーワードリストの利用を考えます。このリストはセッション終了時にSAVE LIST や LIST TO BLOBコマンドで保存され、セッション開始時にLoad list や BLOB to list コマンドで再度読み込まれます。このリストはフローティングパレットに表示され、ユーザがクリックすると最前面のプロセスの選択されたエリアに項目テキ ストが挿入されます。ユーザはドラッグ＆ドロップも使用できます。いずれの場合も重要なことは選択された項目のみを扱うということです。クリックの場合はGET LIST ITEMコマンドのitemPos引数に * を指定して、ドラッグ＆ドロップの場合はDRAG AND DROP PROPERTIES コマンドから返される位置情報をGET LIST ITEM コマンドのitemPos引数に指定して、項目テキストを取得できます。この場合、個々の項目を識別する必要がないため、リスト構築の際itemRef 引数に0以外の任意の数値を渡すことができます。
2. 部分的にリスト項目を識別する必要がある場合 (中級者レベル).

   項目参照番号に、項目を処理する際に必要となる情報を格納することができます。この例はAPPEND TO LIST コマンドの例題で説明しています。この例題では、項目参照番号にレコード番号を格納しています。あわせて[Department]レコード由来の項目と[Employees]レコード由来の項目を区別する必要があり、この点も説明されています。
3. すべての項目リストを個々に識別する必要がある場合 (上級レベル)

   リストの全レベルにおいて、個々の項目を識別する必要のある複雑な階層リスト管理プログラムを作成する必要があるとします。これを実装する簡単な方法は独自のカウンタを使用することです。New list コマンドを使用してhListリストを作成するとします。ここでvhlCounter変数を1に初期化します。APPEND TO LIST や INSERT IN LISTを呼び出すたびにこのカウンタをインクリメントし (vhlCounter:=vhlCounter+1)、カウンタ値を項目参照番号に設定します。項目を削除する場合でもカウンタをデクリメントしないことが重要です。つまりカウンタは増え続けるのみです。この 方法でユニークな項目参照番号を保証できます。番号は倍長整数型なので、20億以上の項目をリストに追加したり挿入したりできます (こんなにも多くのデータを扱う際にはリストではなくテーブルを使用したほうが良いですが)。

Note:ビットワイズ演算子を使用して、項目参照番号に情報を格納できます。例えば2つの整数値、4バイトの値、32個のブール値です。

どのような場合にユニークな参照番号が必要ですか?

階層リストをユーザインタフェースとして使用し、選択された項目のみを処理する場合は、ほとんどの場合項目参照番号を必要としません。Selected list items や GET LIST ITEM を使用すれば、現在選択されている項目を扱うことができます。さらにINSERT IN LIST や DELETE FROM LIST などのコマンドは、選択された項目からの相対位置でリストを操作できます。

基本的に、選択に関係なく、プログラムで任意のリスト項目にアクセスする必要がある場合に項目参照番号が必要です。

APPEND TO LIST

APPEND TO LIST (list ; itemText ; itemRef {; sublist ; expanded})

引数	型		説明
list	ListRef	→	リスト参照番号
itemText	文字	→	新規リスト項目のテキスト
itemRef	倍長整数	→	新規リスト項目の参照番号
sublist	ListRef	→	新規リスト項目に付属するオプションのサブリスト
expanded	ブール	→	オプションのサブリストの展開/折りたたみ

説明

APPEND TO LIST コマンドは、listに渡した参照番号を持つ階層リストに新規項目を追加します。

itemTextには項目テキストを渡します。20億文字までのテキスト式を渡すことができます。

itemRefにはユニークな項目参照番号 (倍長整数型) を渡します。この項目の参照番号はユニークな番号としましたが、実際にはどのような値でも渡すことができます。詳細についてはを参照してください。

項目にサブ項目を設定するには、sublistにサブ階層リストの有効なリスト参照を渡します。この場合、expanded 引数を渡さなければなりません。この引数にTrue または Falseを渡すと、それに応じてサブリストが展開あるいは折りたたまれて表示されます。

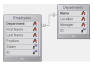
sublistに渡されたリスト参照は既存のリストを参照しなければなりません。既存のリストは1階層あるいはサブリストを持つリストのいずれでも構いません。新規項目にサブリストを添付しない場合は、この引数を省略するか0を渡します。サブリストを添付する場合、sublist と expandedは両方とも渡さなければなりません。

Tips:

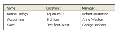
- リストに新規項目を挿入するには、INSERT IN LISTコマンドを使用します。既存の項目のテキストや、そのサブリスト、展開または縮小状態の変更を行うには、SET LIST ITEMコマンドを使用します。
- 新たに追加された項目のアピアランスを変更するには、SET LIST ITEM PROPERTIESコマンドを使用します。

例題

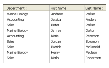
以下は、データベースストラクチャの一部です:



[Departments] と [Employees] テーブルには以下のレコードが含まれています:



Name :	Location :	Manager :
Marine Biology	Aquarium B	Robert Masterson
Accounting	3rd floor	Anne Weston
Sales	First floor-West	George Jackson



Department :	First Name :	Last Name :
Marine Biology	Andrew	Parker
Accounting	Jessica	Anders
Sales	Peter	Parker
Marine Biology	Jeffrey	Dalton
Accounting	Maria	Peterson
Sales	Jordan	Solomon
Sales	Patrick	McDonald
Marine Biology	Henry	Paulson
Sales	Marlo	Robertson

ここで、hlListという名前の階層リストを表示します。この階層リストは、部門を表示するとともに、各部門についてその部門で働いている従業員のサブリストを表示します。hlListのオブジェクトメソッドは以下のようになります:

```
  ` hlList 階層リストオブジェクトメソッド

C_LONGINT($event_l)
C_LONGINT(hlList;$hSubList;$vlDepartment;$vlEmployee;$vlDepartmentID)

$event_l:=Form event

Case of

   :($event_l=On Load)
  ` 空の階層リストを新規に作成
      hlList:=New list
  ` [Departments]テーブルの全レコードを選択
      ALL RECORDS([Departments])
  ` 部門毎に
      For($vlDepartment;1;Records in selection([Departments]))
  ` この部門の従業員を選択
         RELATE MANY([Departments]Name)
  ` 何人いるか
         $vlNbEmployees:=Records in selection([Employees])
  ` 部門に最低１人いるか
         If($vlNbEmployees>0)
  ` 部門項目の子リストを作成
            $hSubList:=New list
  ` 従業員毎に
            For($vlEmployee;1;Records in selection([Employees]))
  ` サブリストに従業員を追加
  ` [Employees]レコードのIDフィールドを
  ` 項目参照番号として使用する
               APPEND TO LIST($hSubList;[Employees]Last Name+", "+
               [Employees]First Name;[Employees]ID)
  ` 次の[Employees]レコードへ
               NEXT RECORD([Employees])
            End for
         Else
  ` 従業員がいない場合、部門項目に子リストを追加しない
            $hSubList:=0
         End if
  ` メインリストに部門項目を追加
  ` [Departments]レコードのIDフィールドを
  ` 項目参照番号として使用する。The bit #31
  ` トップビットを1に設定することで部門と従業員を
  ` 見分けることができる。詳細は後述の説明を参照
         APPEND TO LIST(hlList;[Departments]Name;
         [Departments]ID?+31;$hSublist;$hSubList#0)
 `階層を強調するために 部門項目を太字にする
         SET LIST ITEM PROPERTIES(hlList;0;False;Bold;0)
  ` 次の部門レコード
         NEXT RECORD([Departments])
      End for
  ` リスト全体を昇順にソート
      SORT LIST(hlList;>)
  ` Windowsスタイルを使用してリストを表示
  ` 行の最低の高さを14ポイントにする
      SET LIST PROPERTIES(hlList;Ala Windows;Windows node;14)

   :($event_l=On Unload)
  ` リストはもう必要ない。メモリの解放を忘れずに
      CLEAR LIST(hlList;*)

   :($event_l=On Double Clicked)
  ` ダブルクリックされた
  ` 選択された項目の位置を取得
      $vlItemPos:=Selected list items(hlList)
  ` 念のため位置を確認
      If($vlItemPos#0)
  ` リスト項目の情報を取得
         GET LIST ITEM(hlList;$vlItemPos;$vlItemRef;$vsItemText;$vlItemSubList;$vbItemSubExpanded)
 `項目は部署か？ ?
         If($vlItemRef??31)
  ` そうであれば部署項目がダブルクリックされた
            ALERT("部署項目 "+Char(34)+$vsItemText+Char(34)+" をダブルクリックしました。")
         Else
  ` そうでなければ従業員項目がダブルクリックされた
  ` 親項目IDを使用して[Departments]レコードを検索できる
            $vlDepartmentID:=List item parent(hlList;$vlItemRef)?-31
            QUERY([Departments];[Departments]ID=$vlDepartmentID)
  ` 従業員が働いている部署と上司を表示する
            ALERT("従業員 "+Char(34)+$vsItemText+Char(34)+" をダブルクリックしました。"
            "この人は部署 "+Char(34)+[Departments]Name+Char(34)+
            " で働いていて上司は "+Char(34)+[Departments]Manager+Char(34)+" です。")
         End if
      End if

End case

  ` Note: 4Dはテーブル毎に10億のレコードを格納できます。
  ` この例題では使用されない31番目のビットを用いて
  ` 従業員と部署を区別しています。
```

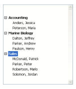
この例では、[Departments]項目と[Employees]項目を区別する理由は1つだけです:

1. レコードIDを項目参照番号に格納しており、[Departments]項目は[Employees]項目と同じ項目参照番号を持つ可能性があります。

2. List item parentコマンドを使用して、選択した項目の親項目を取得しています。対応するレコードIDが10の[Employees]項目をクリックし、同じレコードIDを持つ [Departments]項目も存在する場合、項目参照番号を渡して項目を検索すると、List item parentコマンドは[Departments]項目を見つけます。つまりこのコマンドは[Employees]項目の親ではなく、[Departments]項目の親を返します。

そのようなわけで一意の項目参照番号を作成しましたが、これはユニークな項目参照番号が必要なのではなく、[Departments]と[Employees]レコードを区別する必要があったからです。

フォームを実行すると、リストは以下のように表示されます:



⊟ **Accounting**
Anders, Jessica
Peterson, Maria
⊟ **Marine Biology**
Dalton, Jeffrey
Parker, Andrew
Paulson, Henry
⊟ **Sales**
McDonald, Patrick
Parker, Peter
Robertson, Marlo
Solomon, Jordan

Note: 上記の例は、比較的少ないレコードを扱う場合には、ユーザインターフェイスとして役立ちます。リストはメモリに置かれるので、何百万という項目を持つ階層リストでユーザインターフェイスを作成すべきではありません。

CLEAR LIST

CLEAR LIST (list {; *})
list ListRef ➡ リスト参照番号
* ➡ 指定した場合, サブリストがあればそれもメモリからクリア 省略した場合, サブリストがあってもそれをクリアしない

説明

CLEAR LIST コマンドは、listに渡したリスト参照番号を持つ階層リストを廃棄します。

通常は、リストの項目またはサブ項目にサブリストがあれば一緒にクリアされるように、オプションの引数 * を渡します。

プロパティリストでフォームオブジェクトに関連付けたリストをクリアする必要はありません。そうしたリストのロードとクリアは4Dが自動的に処理します。一方で、BLOBからリストのロード、コピー、取り出しを行ったり、プログラムでリストを作成した場合は、リストを使い終わるたびにCLEAR LISTコマンドを呼ぶ必要があります。

フォームに表示されているリストから項目にあるサブリストだけをクリアするには、以下のように行います:

1. 親項目に対してGET LIST ITEMコマンドを呼び出して、サブリストのリスト参照を取得する。
2. 親項目に対してSET LIST ITEMコマンドを呼び出して、サブリストを消去する前にサブリストを親項目から切り離す。
3. CLEAR LISTコマンドを呼び出して、GET LIST ITEMコマンドで取得した参照番号を持つサブリストを消去する。

例題 1

(ウィンドウを閉じてフォームがアンロードされるときなどに) 必要のないオブジェクトやデータをすべて消去するクリーンアップルーチン中で、クリアしようとするリストはすでにクリアされたものであるかもしれません。Is a listを使用して、クリアする必要のあるリストだけをクリアすることができます:

```
  ` Extract of clean-up routine
 If(Is a list(hlList))
    CLEAR LIST(hlList;*)
 End if
```

例題 2

Load list コマンドの例題参照

例題 3

BLOB to list コマンドの例題参照

Copy list

Copy list (list) -> 戻り値

引数	型	説明
list	ListRef →	コピーするリストの参照
戻り値	ListRef ⮐	複製されたリストのリスト参照番号

説明

Copy list コマンドはlistに渡されたリスト参照番号を持つリストの複製を作成し、新しいリストのリスト参照番号を返します。

このリストの使用を終えたら、CLEAR LISTを呼び出してリストを削除します。

Count list items

Count list items ({* ;} list {; *}) -> 戻り値

引数	型		説明
*	演算子	→	渡された場合, listはオブジェクト名 (文字列) 省略すると, listはリスト参照番号
list	ListRef, 文字	→	リスト参照番号 (* が省略された場合), または リストタイプオブジェクト名 (* が渡された場合)
*	演算子	→	省略すると (デフォルト): 表示されたリスト項目を返す (展開) 指定すると: すべてのリスト項目を返す
戻り値	倍長整数	↩	(展開されて) 表示中のリスト項目数 (2番目の * が省略された場合) またはリスト項目の総数 (2番目の * が指定された場合)

説明

Count list items コマンドは、listに渡した参照番号またはオブジェクト名のリスト上で現在表示中または項目総数を返します。

最初のオプション引数 * を渡すと、list 引数はフォーム中に表示されているリストに対応するオブジェクト名 (文字列) です。この引数を渡さないと、list 引数は階層リスト参照 (ListRef) です。ひとつのリストオブジェクトしか使わない場合や、(2番目の * を渡して) すべての項目の処理を行う場合、どちらのシンタックスも使用できます。他方同じリストを複数のオブジェクトで使用し、(2番目の * を省略して) 表示された項目で処理を行う際は、オブジェクト名に基づくシンタックスを使用する必要があります。それぞれのオブジェクトごとに個別の展開/折りたたみ状態を持つことができるからです。

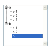
注: リストオブジェクト名に@文字を使用し、フォーム上にこの名前に一致する複数のリストが存在する場合、**Count list items** コマンドは対応する名前の最初のオブジェクトにのみ適用されます。

2番目の * 引数を使用して、返される情報のタイプを指定します。この引数を渡した場合、リストが展開されているか、折りたたまれているかに関わらず、関数は項目総数を返します。
この引数を省略すると、関数はリストやそのサブリストの現在の展開/折りたたみの状態に応じて、表示されている項目数を返します。

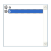
このコマンドは、フォームに表示されているリストで使用します。

例題

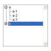
以下はアプリケーションモードで表示された、hListという名前の階層リストです:



```
$vlNbItems:=Count list items(hList) ` この時点で$vlNbItems は8を得ます。
$vlNbTItems:=Count list items(hList;*) `$vlNbTItemsも8を得ます。
```



```
$vlNbItems:=Count list items(hList) ` この時点で$vlNbItems は$vlNbItemsは2を得ます。
$vlNbTItems:=Count list items(hList;*) `$vlNbTItemsは8のままです。
```



```
$vlNbItems:=Count list items(hList) ` この時点で$vlNbItems は$vlNbItemsは5を得ます。
$vlNbTItems:=Count list items(hList;*) `$vlNbTItemsは8のままです。
```

DELETE FROM LIST

DELETE FROM LIST ({* ;} list ; itemRef | * {; *})

*	演算子	→	指定した場合, listはオブジェクト名 (文字列) 省略した場合, listはリスト参照番号
list	ListRef, 文字	→	リスト参照番号 (* が省略された場合), または リストタイプオブジェクト名 (* を渡した場合)
itemRef | *	倍長整数, 演算子	→	項目参照番号, または 0 はリストに最後に追加された項目 または * 現在選択されているリスト項目
*		→	指定した場合, サブリストがあればそれもメモリから消去 省略した場合, サブリストがあってもそれを消去しない

説明

DELETE FROM LIST コマンドは、listに指定した参照番号またはオブジェクト名を持つリストから、itemRef引数で指定した項目を削除します。

1番目のオプション引数 * を渡すと、list 引数はフォーム中のリストオブジェクトのオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数には階層リスト参照 (ListRef) を渡します。階層リストのフォームオブジェクトを1つしか使用しない場合や、2番目の * を省略して階層リスト構造にアクセスする場合は、両方のシンタックスを使用できます。他方、1つの階層リストを複数のフォームオブジェクトで使用する場合や、2番目の * を渡してカレントの項目を処理対象とする場合は、それぞれのオブジェクトが個別にカレント項目を持つため、オブジェクト名に基づくシンタックスを使用しなければなりません。

itemRefに * を渡すと、現在選択されている項目をリストから削除します。この引数に0を渡すとリストに最後に追加された項目が削除されます。

また削除する項目の項目参照番号を指定することができます。指定された項目参照番号を持つ項目が見つからなければ、コマンドは何も行いません。

項目参照番号を指定する場合、それぞれの項目にユニークな参照番号を持つリストをビルドします。そうでなければ項目を識別できません。詳細は階層リストの管理の説明を参照してください。

どの項目を削除するかに関わらず、オプションの * 引数を渡して、4Dがサブリストも削除するように指示すべきです。* 引数を渡さない場合、サブリストのリスト参照番号を事前に取得します。後でCLEAR LIST コマンドを使用してこのリストを削除できます。

例題

以下のコードは現在選択されている項目をhListリストから削除します。項目にサブリストが添付されていれば、そのサブリストおよびさらにそのサブリストも削除されます:

```
DELETE FROM LIST(hList;*;*)
```

Find in list

Find in list ({* ;} list ; value ; scope {; itemsArray {; *}}) -> 戻り値

引数	型		説明
*	演算子	→	指定した場合, listはオブジェクト名 (文字列) 省略した場合, listはリスト参照番号
list	ListRef, 文字	→	リスト参照番号 (* 省略時) リストオブジェクト名 (* 指定時)
value	文字	→	検索する値
scope	整数	→	0=メインリスト, 1=サブリスト
itemsArray	倍長整数配列	←	2番目の * 省略時: 見つけた項目の位置配列 - 2番目の * 指定時: 見つけた項目の項目参照番号配列
*	演算子	→	- 省略時: 項目位置を使用 - 指定時: 項目参照番号使用
戻り値	倍長整数	↩	- 2番目の * 省略時: 見つけた項目の位置 - 2番目の * 指定時: 見つけた項目の項目参照番号

説明

Find in list コマンドは、valueに渡した文字列と同じ値を持つ項目をlistリスト中で検索し、最初に見つけた項目の位置または項目参照番号を返します。複数の項目を見つけた場合、コマンドは itemsArray 配列にそれらの一または項目参照番号を返します。

1番目の * 引数を渡した場合、list 引数はフォーム中のリストオブジェクトのオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数はリスト参照番号 (ListRef) です。1つしかフォーム上でリストオブジェクトを使用しない場合や、2番目の * を渡して項目参照番号を扱う場合は、いずれのシンタックスも使用できます。対して複数のリストオブジェクトがフォーム上にあり、2番目の * を省略して項目位置を扱う場合は、リストオブジェクトごとに項目位置が異なるため、オブジェクト名に基づくシンタックスを使用しなければなりません。

Note: オブジェクト名に@文字を使用し、フォーム上にこれに合致するリストオブジェクトが複数ある場合、Find in list コマンドは最初に見つけたリストオブジェクトを検索の対象とします。

2番目の * 引数は項目の位置を取得するか (この引数を省略時)、項目参照を取得するか (この引数指定時) を指示します。

valueには検索する文字列を渡します。検索は完全一致で行います。つまり"wood"を検索すると"wooden"は見つかりません。しかしワイルドカード文字を使用して (@) 前方一致や後方一致、含む検索などを行うことができます。

scope 引数を使用して、検索対象をリストのトップレベルに限定するか、サブリストを含めるか指定できます。リストの第一レベルに限定するには0を渡します。1を渡すとサブリスト内も検索されます。

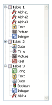
valueに一致するすべての位置あるいは参照番号を取得したい場合、倍長整数配列をオプションのitemsArray 引数に渡します。必要であればコマンドが配列を作成しリサイズします。コマンドは見つけた項目の、2番目の * が省略されていれば位置を、省略されていなければ項目参照番号を返します。

位置はメインリストの先頭項目からの相対位置です。その時点でのサブリストの展開/折りたたみ状況が考慮されます。

value に対応する値が見つからない場合、コマンドは0を返し、itemsArray配列は空になります。

例題

以下の階層リストにおいて:



```
$vlItemPos:=Find in list(hList;"P@";1;$arrPos)
  `$vlItemPos は 6
  `$arrPos{1} は 6 で $arrPos{2} は 11
$vlItemRef:=Find in list(hList;"P@";1;$arrRefs;*)
  `$vlItemRef は 7
  `$arrRefs{1} は 7 で $arrRefs{2} は 18
$vlItemPos:=Find in list(hList;"Date";1;$arrPos)
  `$vlItemPos は9
  `$arrPos{1} は9 で$arrPos{2} は16
$vlItemRefFind in list(hList;"Date";1;$arrRefs;*)
  `$vlItemRef は11
  `$arrRefs{1} は11 で$arrRefs{2} は23
$vlItemPos:=(hList;"Date";0;*)
  `$vlItemPos は0
```

GET LIST ITEM

GET LIST ITEM ({* ;} list ; itemPos | * ; itemRef ; itemText {; sublist ; expanded})

引数	型		説明
*	演算子	→	指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→	リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
itemPos | *	演算子, 倍長整数	→	展開/折りたたまれたリスト中の項目位置 * の場合、リスト中のカレント項目
itemRef	倍長整数	←	項目参照番号
itemText	文字	←	リスト項目テキスト
sublist	ListRef	←	サブリストリスト参照番号 (サブリストがある場合)
expanded	ブール	←	サブリストが添付されている場合: TRUE = サブリストは現在展開されている FALSE = サブリストは折りたたまれている

説明

GET LIST ITEM コマンドは、リスト参照番号またはオブジェクト名がlistであるリスト中、itemPosで指定した項目に関する情報を返します。

1番目の * 引数を渡した場合、list 引数はフォーム中のリストオブジェクトのオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数はリスト参照番号 (ListRef) です。1つしかフォーム上でリストオブジェクトを使用しない場合、いずれのシンタックスも使用できます。同じリストの複数のリストオブジェクトがフォーム上にある場合は、リストオブジェクトごとに展開/折りたたみが異なるため、オブジェクト名に基づくシンタックスを使用しなければなりません。

Note: オブジェクト名に@文字を使用し、フォーム上にこれに合致するリストオブジェクトが複数ある場合、GET LIST ITEM コマンドは最初に見つけたリストオブジェクトを検索の対象とします。

項目位置は、リストの現在の展開/折りたたみ状況に基づき、相対的に示されなければなりません。1 から Count list itemsで返される値までの数値を渡します。この範囲外の数値を渡すと、GET LIST ITEMは空の値 (0, "", など) を返します。

コマンドの呼出し後、以下の情報が返されます:

- itemRefに項目参照番号。
- itemTextに項目テキスト。

オプションの引数 sublist と expanded を渡した場合:

- subListに、その項目に添付されたサブリストのリスト参照番号。項目にサブリストが添付されていない場合、subListには0が返されます。
- 項目にサブリストが添付されているとき、expandedにはサブリストが展開されていればTRUEが、折りたたまれていればFALSEが返されます。

例題 1

hList は項目にユニークな参照番号が与えられたリストです。以下のコードはプログラムで、現在選択されている項目のサブリストの展開/折りたたみを切り替えます:

```
$vlItemPos:=Selected list items(hList)
If($vlItemPos>0)
   GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText;$hSublist;$vbExpanded)
   If(Is a list($hSublist))
      SET LIST ITEM(hList;$vlItemRef;$vsItemText;$vlItemRef;$hSublist;Not($vbExpanded))
   End if
End if
```

例題 2

APPEND TO LIST コマンドの例題を参照

Get list item font

Get list item font ({* ;} list ; itemRef | *) -> 戻り値

引数	型	説明
*	演算子	→ 指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→ リスト参照番号 (* 省略時) または リストオブジェクト名 (* 指定時)
itemRef | *	倍長整数, 演算子	→ 項目参照番号 または 0 の場合最後に追加された項目 または *の場合リストのカレント項目
戻り値	文字	↩ フォント名

説明

Get list item font コマンドは、リスト参照またはオブジェクト名で指定したlistリストの、itemRef 引数で指定した項目のフォント名を返します。

一番目のオプション引数 * を渡すと、list 引数はフォーム上のリストオブジェクトのオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数は階層リスト参照 (ListRef) です。フォーム上に1つしかリストオブジェクトがない場合や (2番目の * を省略して) リストのストラクチャを対象に処理を行う場合、いずれのシンタックスも使用できます。同じリストを複数のリストオブジェクトで使用する場合で、(2番目の * を渡して) 選択されている項目を対象に処理を行う場合、選択されている項目はオブジェクトごとに異なるので、オブジェクト名のシンタックスを使用しなければなりません。

Note: オブジェクト名に@文字を使用し、フォーム上にこれに合致するリストオブジェクトが複数ある場合、Get list item font コマンドは最初に見つけたリストオブジェクトを検索の対象とします。

itemRefには項目参照番号を渡すことができます。この番号に対応する項目がない場合、コマンドは何も行いません。itemRefに0を渡すと(APPEND TO LISTを使用して) リストに最後に追加された項目のフォントを返します。
itemRefに * を渡した場合、コマンドはリスト中のカレントの項目のフォントを返します。複数の項目がユーザにより選択されている場合、最後に選択された項目がカレントの項目です。項目が選択されていない場合、コマンドは何も行いません。

GET LIST ITEM ICON

GET LIST ITEM ICON ({* ;} list ; itemRef | * ; icon)

引数	型	説明
*	演算子	⇒指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	⇒リスト参照番号 (* 省略時) または リストオブジェクト名 (* 指定時)
itemRef | *	演算子, 倍長整数	⇒項目参照番号 または 0: リストに最後に追加された項目 または *: リストのカレントの項目
icon	ピクチャー変数	⇐項目に関連付けられたアイコン

説明

GET LIST ITEM ICON コマンドは、listに参照番号またはオブジェクト名を渡したリスト内の、itemRef項目参照の項目に割り当てられたアイコンをiconに返します。

オプションの第一引数 * を渡すと、list 引数はフォーム上のリストオブジェクトに対応するオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数は階層リスト参照 (ListRef) です。リストオブジェクトを一つしか使わない場合や、2番目の * を使用しない場合は、両方のシンタックスを使用できます。他方フォーム上に同じ階層リストを参照する複数のオブジェクトがある場合で、2番目の * を渡して現在選択されている項目を参照する場合、それぞれのオブジェクトが個別に選択された項目をもつので、オブジェクト名に基づくシンタックスを使用しなければなりません。

Note: オブジェクト名に @ 文字を使用することで、名前に対応するオブジェクトが複数検索された場合、GET LIST ITEM ICON コマンドは最初に見つけたオブジェクトを処理の対象とします。

itemRefに項目参照番号を渡すことができます。対応する項目がない場合、コマンドは何も行いません。0を渡した場合、リストに最後に追加された項目が処理の対象となります。* を渡した場合、コマンドは現在選択されている項目を処理の対象とします。複数の項目が選択されている場合、カレントの項目は最後に選択された項目です。項目が選択されていない場合、コマンドは何も行いません。

iconにはピクチャ変数を渡します。コマンド実行後、(スタティックピクチャ、リソース、ピクチャ式になどの) ソースに関わらず、引数には項目に割り当てられたアイコンが返されます。

項目にアイコンが割り当てられていない場合、icon変数は空となります。

Note: 項目に割り当てられたアイコンがスタティックな参照 (リソース参照またはピクチャライブラリのピクチャ) で定義されている場合、その番号はGET LIST ITEM PROPERTIES コマンドで取得できます。

GET LIST ITEM PARAMETER

GET LIST ITEM PARAMETER ({* ;} list ; itemRef | * ; selector ; value)

引数	型		説明
*	演算子	→	指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→	リスト参照番号 (* 省略時) または リストオブジェクト名 (* 指定時)
itemRef | *	演算子, 倍長整数	→	項目参照番号 または 0: リストに最後に追加された項目 または *: カレントのリスト項目
selector	文字	→	パラメタ定数
value	文字, ブール, 実数	←	パラメタの現在値

説明

GET LIST ITEM PARAMETER コマンドは、list 引数にリスト参照やオブジェクト名で指定したリストの、itemRef 項目の、selector パラメタに対応する現在値を取得するために使用します。

オプションの第一引数 * を渡すと、list 引数はフォーム上のリストオブジェクトに対応するオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数は階層リスト参照 (ListRef) です。リストオブジェクトを一つしか使わない場合や、2番目の * を使用しない場合は、両方のシンタックスを使用できます。他方フォーム上に同じ階層リストを参照する複数のオブジェクトがある場合で、2番目の * を渡して現在選択されている項目を参照する場合、それぞれのオブジェクトが個別に選択された項目をもつので、オブジェクト名に基づくシンタックスを使用しなければなりません。

Note: オブジェクト名に @ 文字を使用することで、名前に対応するオブジェクトが複数検索された場合、GET LIST ITEM PARAMETER コマンドは最初に見つけたオブジェクトを処理の対象とします。

itemRefに項目参照番号を渡すことができます。対応する項目がない場合、コマンドは何も行いません。0を渡した場合、リストに最後に追加された項目が処理の対象となります。* を渡した場合、コマンドは現在選択されている項目を処理の対象とします。複数の項目が選択されている場合、カレントの項目は最後に選択された項目です。項目が選択されていない場合、コマンドは何も行いません。

selectorには""テーマのAdditional text定数または任意のカスタム値を渡すことができます。selector と value 引数に関する詳細は、SET LIST ITEM PARAMETER コマンドの説明を参照してください。

GET LIST ITEM PARAMETER ARRAYS

GET LIST ITEM PARAMETER ARRAYS ({* ;} list ; itemRef | * ; arrSelection {; arrValues})

引数	型		説明
*	演算子	⇒	指定時listはオブジェクト名 (文字列) 省略時listはリスト参照番号
list	ListRef, 文字	⇒	リストタイプのオブジェクト名 (* 指定時) またはリスト参照番号 (* 省略時)
itemRef | *	倍長整数, 演算子	⇒	項目参照番号、または 0 = リストに最後に追加された項目、または * = カレントリスト項目
arrSelection	テキスト配列	⇐	パラメーター名配列
arrValues	テキスト配列	⇐	パラメーター値配列

説明

GET LIST ITEM PARAMETER ARRAYSコマンドはlist 引数で指定した参照またはオブジェクト名を持つ階層リスト中で、itemRef で指定した項目に割り当てられたパラメーター (およびその値) を一回の呼び出しで取り出すことができます。

項目に関連付けられたパラメーターは各項目の追加の情報を格納しています。これらはSET LIST ITEM PARAMETERコマンドで設定できます。

一番目のオプションの引数 * を渡すと、list はフォーム中でリストを表示するリストオブジェクトに対応するオブジェクト名 (文字列) です。この引数を渡さない場合、list は階層リスト参照 (ListRef) です。ひとつのリストオブジェクトだけ、あるいは (二番目の * を使用しないで) 項目を構造的に選択する場合、どちらのシンタックスでも使用できます。しかし同じリストを複数のリストオブジェクトに表示し、かつ (二番目の * を使用して) 現在選択されている項目に対してコマンドを適用する場合、各リストオブジェクトは異なるカレント項目を持つことができるため、オブジェクト名に基づくシンタックスを使用しなければなりません。

GET LIST ITEM PARAMETER ARRAYSはitemRef項目に設定されたパラメーターをarrSelectorsテキスト配列に返します。arrValuesテキスト配列が渡された場合、これらのパラメーターに設定された値も返します。

arrValues はテキスト配列でなければなりません。関連付けた値がテキストでない (数値やブール) の場合、文字に変換されます (True="1", False="0")。

例題

以下のコードで構築された階層リストがあるとき:

```
<>HL:=New list
$ID:=30
APPEND TO LIST(<>HL;"Martin";$ID)
 //5つのパラメーター
SET LIST ITEM PARAMETER(<>HL;$ID;"Firstname";"Phil")
SET LIST ITEM PARAMETER(<>HL;$ID;"Birthday";"01/02/1978")
SET LIST ITEM PARAMETER(<>HL;$ID;"Male";True) //ブール
SET LIST ITEM PARAMETER(<>HL;$ID;"Age";33) //数値
SET LIST ITEM PARAMETER(<>HL;$ID;"City";"Dallas")
```

*説明を簡単にするために、階層リスト<>HLが同じ名前のフォームオブジェクト"<>HL"に関連づけられています。
リスト中で"Martin"が選択されているとき、以下のコードを使用してそのパラメーターを取得できます:

```
ARRAY TEXT(arrParamNames;0)
GET LIST ITEM PARAMETER ARRAYS(*;"<>HL";*;arrParamNames)
 // arrParamNames{1} = "Firstname"
 // arrParamNames{2} = "Birthday"
 // arrParamNames{3} = "Male"
 // arrParamNames{4} = "Age"
 // arrParamNames{5} = "City"
```

パラメーターの値も取得したい場合、以下のようにします:

```
ARRAY TEXT(arrParamNames;0)
ARRAY TEXT(arrParamValues;0)
GET LIST ITEM PARAMETER ARRAYS(*;"<>HL";*;arrParamNames;arrParamValues)
 // arrParamValues{1} = "Phil"
 // arrParamValues{2} = "01/02/1978"
 // arrParamValues{3} = "1"
 // arrParamValues{4} = "33"
 // arrParamValues{5} = "Dallas"
```

GET LIST ITEM PROPERTIES

GET LIST ITEM PROPERTIES ({* ;} list ; itemRef | * ; enterable {; styles {; icon {; color}}})

引数	型	説明
*	演算子	→ 指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→ リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
itemRef | *	演算子, 倍長整数	→ 項目参照番号, または 0: リストに最後に追加された項目, または *: カレントのリスト項目
enterable	ブール	← TRUE = 入力可, FALSE = 入力不可
styles	倍長整数	← 項目のフォントスタイル
icon	倍長整数	← "cicn" Mac OSベースのリソースID, または 65536 + "PICT" Mac OSベースのリソースID, または 131072 + ピクチャ参照番号
color	倍長整数	← RGBカラー値

説明

GET LIST ITEM PROPERTIES コマンドは、引数listに渡されたリスト参照番号またはオブジェクト名のリスト内で、引数itemRefによって指定された項目のプロパティを返します。

オプションの第一引数 * を渡すと、list 引数はフォーム上のリストオブジェクトに対応するオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数は階層リスト参照 (ListRef) です。リストオブジェクトを一つしか使わない場合や、2番目の * を使用しない場合は、両方のシンタックスを使用できます。他方フォーム上に同じ階層リストを参照する複数のオブジェクトがある場合で、2番目の * を渡して現在選択されている項目を参照する場合、それぞれのオブジェクトが個別に選択された項目をもつので、オブジェクト名に基づくシンタックスを使用しなければなりません。

Note: オブジェクト名に @ 文字を使用することで、名前に対応するオブジェクトが複数検索された場合、GET LIST ITEM PROPERTIES コマンドは最初に見つけたオブジェクトを処理の対象とします。

itemRefに項目参照番号を渡すことができます。0を渡した場合、リストに最後に追加された項目が処理の対象となります。* を渡した場合、コマンドは現在選択されている項目を処理の対象とします。複数の項目が選択されている場合、カレントの項目は最後に選択された項目です。
* を渡して項目が選択されていない場合や項目参照番号を渡してその項目が存在しない場合、コマンドはパラメタを変更しません。

項目参照番号を使用して作業を実行する場合、項目がユニークな参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細についてはの説明を参照してください。

呼出し後に、以下の値を取得できます:

- 項目が編集可の場合、enterableにTRUEが返されます。
- stylesには項目のフォントスタイルが返されます。
- iconには項目に割り当てられたアイコンまたはピクチャが返されます。アイコンがない場合0が返されます。
- colorには項目のテキストカラーが返されます。

Note: GET LIST ITEM ICON コマンドを使用すれば、ピクチャ変数にアイコンを取得できます。

これらのプロパティに関する詳細は、SET LIST ITEM PROPERTIESコマンドの説明を参照してください。

GET LIST PROPERTIES

GET LIST PROPERTIES (list ; appearance {; icon {; lineHeight {; doubleClick {; multiSelections {; editable}}}}})

引数	型		説明
list	ListRef	→	リスト参照番号
appearance	倍長整数	←	リストの描画スタイル 1 = Macスタイル 2 = Windowsスタイル
icon	倍長整数	←	"cicn" Mac OSベースのリソースID
lineHeight	倍長整数	←	行の最小高さ (ピクセル単位)
doubleClick	倍長整数	←	ダブルクリックでサブリストを展開/折り畳み 0 = Yes, 1= No
multiSelections	倍長整数	←	複数行選択: 0 = No, 1 = Yes
editable	倍長整数	←	ユーザによる更新可: 0 = No, 1 = Yes

説明

GET LIST PROPERTIES コマンドは、listで指定された参照番号を持つリストについての情報を返します。

引数 appearanceは、リストの表示形式を返します。

引数 iconは、リストに表示されるノードアイコンのリソースIDを返します。

引数 lineHeightは、行の高さの最小値を返します。

引数 doubleClickが1に設定されている場合、親リスト項目をダブルクリックしても子リストが開いたり閉じたりしません。0に設定されているときは開閉の動作をします（デフォルト値）。

引数 multiSelectionsに0が代入された場合、そのリストにおいてリスト項目を複数選択することはできません（手動でもプログラムからでも）。1が代入された場合は、複数項目を選択することができます。

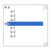
引数 editableに1が代入された場合、レコードの選択リストとして表示されると、そのリストは編集可能になります。0が代入された場合、リストを編集することはできません。

これらのプロパティは、SET LIST PROPERTIESコマンドおよび、リストがデザインモードのリストエディタで作成された場合、またはSAVE LISTコマンドを使用して保存された場合は、リストエディタで設定することができます。

リストの表示様式、ノードアイコン、行の高さの最小値、およびダブルクリックの管理についての詳細は、SET LIST PROPERTIESコマンドを参照してください。

例題

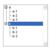
次に示すhListという名前のリストがアプリケーションモードにあるとします:



以下はボタンのオブジェクトメソッドです:

```
  ` bMacOrWin button Object Method
 GET LIST PROPERTIES(hList;$vlAppearance;$vlIcon;$vlLH;$vlClick;$vlSelect;$vlModif)
 If($vlAppearance=Ala Macintosh)
    $vlAppearance:=Ala Windows
    $vlIcon:=Windows node
    $vlModif:=1
 Else
    $vlAppearance:=A la Macintosh
    $vlIcon:=Macintosh node
    $vlModif:=1
 End if
 SET LIST PROPERTIES(hList;$vlAppearance;$vlIcon;$vlLH;$vlClick;$vlSelect;$vlModif)
```

このメソッドにより、以下のように表示されます:



INSERT IN LIST

INSERT IN LIST ({* ;} list ; beforeItemRef | * ; itemText ; itemRef {; sublist ; expanded})

引数	型	説明
*	演算子	⇨指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	⇨リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
beforeItemRef | *	倍長整数, 演算子	⇨項目参照番号 または 0: リストに最後に追加された項目 または *: 現在選択されている項目
itemText	文字	⇨新しいリスト項目のテキスト
itemRef	倍長整数	⇨新しいリスト項目のユニークな参照番号
sublist	ListRef	⇨新しいリスト項目に添付するオプションのサブリスト
expanded	ブール	⇨サブリストの展開/折りたたみ状態を指定

説明

INSERT IN LIST コマンドは、listに渡されたリスト参照番号またはオブジェクト名のリストに新規項目を挿入します。

オプションの第一引数 * を渡すと、list 引数はフォーム上のリストオブジェクトに対応するオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数は階層リスト参照 (ListRef) です。リストオブジェクトを一つしか使わない場合や、2番目の * を使用しない場合は、両方のシンタックスを使用できます。他方フォーム上に同じ階層リストを参照する複数のオブジェクトがある場合で、2番目の * を渡して現在選択されている項目を参照する場合、それぞれのオブジェクトが個別に選択された項目をもつので、オブジェクト名に基づくシンタックスを使用しなければなりません。

beforeItemRef 引数は、新規リスト項目を挿入する位置を指定するために用います:

- リストに最後追加された項目の前に新しい項目を挿入するには値0を渡します。そして新しく挿入された項目が選択されます。
- 現在選択されている項目の前に新しい項目を挿入するには * を指定します。そして新しく挿入された項目が選択されます。
- 特定の項目の前に挿入するには、その項目の参照番号を渡します。この場合新しく挿入された項目は自動的には選択されません。項目参照番号に対応する項目がない場合、コマンドは何もしません。

新規項目のテキストと項目参照番号をitemTextとitemRefに渡します。

項目にサブ項目を設定する場合は、sublist に有効なリスト参照を渡します。この場合サブリストを展開または縮小を指定するexpanded引数を渡さなければなりません。TrueまたはFalseの指定に従い、サブリストは展開または折りたたまれます。

例題

以下のコードは、リストhListの現在選択されている項目の直前に項目を挿入します (サブリストは添付されていません):

```
vlUniqueRef:=vlUniqueRef+1
INSERT IN LIST(hList;*;"New Item";vlUniqueRef)
```

Is a list

Is a list (list) -> 戻り値

引数	型		説明
list	ListRef	→	テストするListRef値
戻り値	ブール	↩	TRUE: listは階層リスト FALSE: listは階層リストでない

説明

Is a list コマンドは、list 引数で指定された値が階層リストの有効な参照の場合Trueを返します。それ以外の場合Falseを返します。

例題 1

CLEAR LISTコマンドの例題参照

例題 2

DRAG AND DROP PROPERTIESコマンドの例題参照

List item parent

List item parent ({* ;} list ; itemRef | *) -> 戻り値

引数	型		説明
*	演算子	→	指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→	リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
itemRef | *	演算子, 倍長整数	→	項目参照番号, または 0: リストに最後に追加された項目, または *: カレントのリスト項目
戻り値	倍長整数	↩	親項目の項目参照番号 または ない場合0

説明

List item parent コマンドは、親項目の項目参照番号を返します。

listにはリスト参照番号またはオブジェクト名を渡します。

オプションの第一引数 * を渡すと、list 引数はフォーム上のリストオブジェクトに対応するオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数は階層リスト参照 (ListRef) です。リストオブジェクトを一つしか使わない場合や、2番目の * を使用しない場合は、両方のシンタックスを使用できます。他方フォーム上に同じ階層リストを参照する複数のオブジェクトがある場合で、2番目の * を渡して現在選択されている項目を参照する場合、それぞれのオブジェクトが個別に選択された項目をもつので、オブジェクト名に基づくシンタックスを使用しなければなりません。

Note: オブジェクト名に @ 文字を使用することで、名前に対応するオブジェクトが複数検索された場合、List item parent コマンドは最初に見つけたオブジェクトを処理の対象とします。

itemRefにはリスト中の項目参照番号、あるいは0、または * を渡します。0を渡した場合、コマンドはリストに最後に追加した項目に適用されます。* を渡した場合、コマンドはリストのカレント項目に適用されます。複数の項目が選択されている場合、最後に選択された項目がカレントの項目となります。

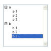
戻り値として、対応する項目がリスト上に存在し、かつその項目がサブリスト上にある場合（つまり親項目を持つ）、親項目の項目参照番号が返されます。

渡した項目参照番号を持つ項目が存在しない場合や、* を渡したが選択項目が存在しない場合、または項目が親を持たない場合は、List item parentは0を返します。

項目参照番号を使用して作業を実行する場合、項目がユニークな参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、APPEND TO LISTの説明を参照してください。

例題

hListがあるとき、アプリケーションモードで以下のように表示されます:



項目参照番号は以下のように設定されています:

項目	項目参照番号
a	100
a - 1	101
a - 2	102
b	200
b - 1	201
b - 2	202
b - 3	203

- 項目"b - 3"が選択されているとき、以下のコードで$vlParentItemRefには200が返されます。これは項目"b"の項目参照番号です:

```
 $vlItemPos:=Selected list items(hList)GET LIST
ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText)$vlParentItemRef:=List item parent(hList;$vlItemRef) `
$vlParentItemRef gets 200
```

- "a - 1"が選択されていると、"a".の項目参照番号である100が$vlParentItemRef に返されます。
- "a" または "b"が選択されていると、これらの項目には親項目がないため$vlParentItemRefには0が返されます。

List item position

List item position ({* ;} list ; itemRef) -> 戻り値

引数	型		説明
*	演算子	→	指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→	リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
itemRef	倍長整数	→	項目参照番号
戻り値	倍長整数	↩	展開/折りたたまれたリストにおける項目位置

説明

List item position コマンドは、listに渡された項目参照番号またはオブジェクト名リスト中で、itemRefで指定した項目の位置を返します。

オプションの第一引数 * を渡すと、list 引数はフォーム上のリストオブジェクトに対応するオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数は階層リスト参照 (ListRef) です。リストオブジェクトを一つしか使わない場合、両方のシンタックスを使用できます。他方フォーム上に同じ階層リストを参照する複数のオブジェクトがある場合、それぞれのオブジェクトが個別に展開/折りたたみ状態をもつので、オブジェクト名に基づくシンタックスを使用しなければなりません。

Note: オブジェクト名に @ 文字を使用することで、名前に対応するオブジェクトが複数検索された場合、List item position コマンドは最初に見つけたオブジェクトを処理の対象とします。

Note: このテーマの他のコマンドと異なり、このコマンドではitemRefに0を渡して最後に追加された項目を指定することはできません。

位置はリストやサブリストの展開/折りたたみ状態を使用し、メインリストの先頭項目からの相対位置であらわされます。

結果は1からCount list itemsで返される数値までの間となります。

項目が縮小されているリストにあるために表示されていない場合、List item positionが適切なリストを展開してその項目を表示します。

項目が存在しない場合、LList item positionは0を返します。

LIST OF CHOICE LISTS

LIST OF CHOICE LISTS (numsArray ; namesArray)

引数	型	説明
numsArray	倍長整数配列	← 選択リスト番号
namesArray	テキスト配列	← 選択リスト名

説明

LIST OF CHOICE LISTS コマンドは同期されたnumsArr と namesArr 配列に、デザインモードのリストエディタで定義された選択リストの番号と名前を返します。

選択リストの番号はそれが作られた順番に対応します。リストエディタ中、リストは名前順に表示されます。

Load list

Load list (listName) -> 戻り値

引数	型	説明
listName	文字	→ デザインモードのリストエディタで 作成されたリスト名
戻り値	ListRef	↩ 新しく作成されたリストのリスト参照番号

説明

Load list は、listNameで指定した名前のリストのコピーを作成し、そのリスト参照番号を返します。

データベース中で、指定されたリストを検索するにはLIST OF CHOICE LISTS コマンドを使用します。
listNameと同じ名前のリストが存在するかを調べるにはIs a listコマンドを使用します。

作成されるリストはデザインモードで定義されたリストのコピーであることに留意してください。したがって、新規リストに変更を加えても、デザインモードで定義されたリストには影響しません。またデザインモードで定義されたリストに変更を加えても、すでにコピー済みのリストに影響しません。

新たに作成したリストを変更し、その変更を恒久的に保存したい場合は、SAVE LISTコマンドを呼び出します。

新たに作成したリストを使い終わったら、CLEAR LISTコマンドを呼び出してそれを廃棄するのを忘れないでください。廃棄しないと、作業セッションが終了するまで、またはそれを作成したプロセスが終了するかアボートされるまで、そのリストはメモリに留まります。

Tip: プロパティリストの選択リストプロパティを使用して、リストをフォームオブジェクト (階層リスト、タブコントロール、階層ポップアップメニュー) に関連付ける場合、オブジェクトメソッドからLoad listやCLEAR LISTを呼び出す必要はありません。4Dはリストを自動的にロードして消去します。

例題

国際市場に対応するデータベースを作成し、そのデータベースの使用中に異なる言語に切り替える必要があるとします。フォームでhlListという名前の、標準オプションのリストを示す階層リストを提供します。デザインモードでは、英語版の"Std Options US"、フランス語版の"Std Options FR"、日本語版の"Std Options JP"等、さまざまなリストを準備しました。これに加えて、<>gsCurrentLanguageという名前のインタープロセス変数を持ち、これに、英語版には"US"、フランス語版には"FR"、日本語版には"JP"というように2文字の言語コードを格納します。現在選択されている言語を使用してリストをロードするために、以下のように記述します:

```
  ` hlList Hierarchical List Object Method
 Case of
    :(Form event=On Load)
       C_LONGINT(hlList)
       hlList:=Load list("Std Options"+<>gsCurrentLanguage)
    :(Form event=On Unload)
       CLEAR LIST(hlList;*)
 End case
```

New list

New list -> 戻り値

引数	型		説明
戻り値	ListRef	↩	リスト参照番号

説明

New list は、新しい空の階層リストをメモリに作成し、ユニークなリスト参照番号を返します。

警告: 階層リストはメモリに保持されます。階層リストを使い終わったら、CLEAR LISTコマンドを使用してそれを廃棄し、メモリを解放することが重要です。

階層リストを作成するコマンドは、この他にもあります:

- Copy list は、既存のリストからリストを複製します。
- Load list は、デザインモードのリストエディタで (手動またはプログラムによって) 作成された選択リストをロードすることによりリストを作成します。
- BLOB to list は、前回保存されたBLOBのコンテンツからリストを作成します。

New listを使用して階層リストを作成した後は、以下のことが行えます:

- APPEND TO LISTやINSERT IN LISTを使用して、項目をリストに追加できます。
- DELETE FROM LISTコマンドを使用して、リストから項目を削除できます。.

例題

APPEND TO LISTコマンドの例題を参照

SAVE LIST

SAVE LIST (list ; listName)

引数	型		説明
list	ListRef	⇒	リスト参照番号
listName	文字	⇒	デザインモードのリストエディタに 登録されるリスト名

説明

SAVE LIST コマンドは、デザインモードのリストエディタに、listに渡した参照番号を持つリストをlistNameに渡した名前で保存します。

既にその名前のリストが存在する場合は、その内容が置き換えられます。

SELECT LIST ITEMS BY POSITION

SELECT LIST ITEMS BY POSITION ({* ;} list ; itemPos {; positionsArray})

引数	型	説明
*	演算子	→ 指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→ リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
itemPos	倍長整数	→ 展開/折りたたまれたリスト中の項目位置
positionsArray	倍長整数配列	→ 展開/折りたたまれたリスト中の項目位置配列

説明

SELECT LIST ITEMS BY POSITION コマンドは、listに渡された参照番号のリストにおいて、itemPosならびにオプションとしてpositionsArrayに渡された位置にある項目を選択します。

オプションの第一引数 * を渡すと、list 引数はフォーム上のリストオブジェクトに対応するオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数は階層リスト参照 (ListRef) です。リストオブジェクトを一つしか使わない場合、両方のシンタックスを使用できます。他方フォーム上に同じ階層リストを参照する複数のオブジェクトがある場合、それぞれのオブジェクトが個別に展開/折りたたみ状態をもつので、オブジェクト名に基づくシンタックスを使用しなければなりません。

Note: オブジェクト名に @ 文字を使用することで、名前に対応するオブジェクトが複数検索された場合、SELECT LIST ITEMS BY POSITION コマンドは最初に見つけたオブジェクトを処理の対象とします。

項目の位置は常に、リストとそのサブリストの展開や折りたたみの現在の状態を使用して表わされます。位置の値として、1からCount list itemsによって返される値までの数値を渡します。この範囲外の値を渡すと、項目は選択されません。

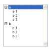
引数positionsArrayを渡さない場合、引数itemPosは選択する項目の位置を示します。

オプションの引数positionsArrayを使用すると、list内の複数の項目を同時に選択することができます。positionsArrayには必ず配列を渡し、その各要素は選択する項目の位置を示します。
この引数を渡すと、itemPos引数に指定した項目が、選択された項目中そのリストの新しいカレント項目になります。この項目は、配列で定義した一連の項目に含まれていない可能性もあります。カレント項目とは、具体的に言うと、EDIT ITEMコマンドを使用した場合に編集モードに移行される項目のことです。

Note: 階層リスト内で複数のリスト項目を同時に選択するには (手動、あるいはプログラムから)、そのリストに対して複数選択可プロパティを有効に設定しておかなくてはなりません。このプロパティの設定は、SET LIST PROPERTIESコマンドを使用して行います。

例題

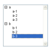
アプリケーションモードで以下のように表示される階層リストhListがあります:



以下のコードを実行すると:

```
SELECT LIST ITEMS BY POSITION(hList;Count list items(hList))
```

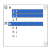
表示されているリスト項目の最後の項目が選択されます:



以下のコードを実行すると:

```
SET LIST PROPERTIES(hList;0;0;18;0;1)
 `It is imperative to pass 1 as the last parameter in order to allow multiple selections
ARRAY LONGINT($arr;3)
$arr{1}:=2
$arr{2}:=3
$arr{3}:=5
SELECT LIST ITEMS BY POSITION(hList;3;$arr)
 `The 3rd item is designated as the current item
```

.. 2,3,5番目の項目が選択されます:



SELECT LIST ITEMS BY REFERENCE

SELECT LIST ITEMS BY REFERENCE (list ; itemRef {; refArray})

引数	型		説明
list	ListRef	→	リスト参照番号
itemRef	倍長整数	→	項目参照番号 または 0: リストに最後に追加された項目
refArray	倍長整数配列	→	項目参照番号配列

説明

SELECT LIST ITEMS BY REFERENCE コマンドは、listに渡された参照番号のリストにおいて、itemRefならびにオプションとしてrefArrayに渡された項目参照番号を持つ項目を選択します。

渡した項目参照番号の項目が存在しない場合、コマンドは何も行いません。

項目が現在表示されていない場合（つまり、縮小されているサブリスト内にその項目がある場合）、コマンドは必要なサブリストを展開し、その項目を表示します。

refArray引数を渡さない場合、itemRef引数は選択される項目の参照番号を示します。項目番号がそのリスト中の項目に対応しない場合、コマンドは何も行いません。リストに最後に追加された項目を示すために値0を指定することもできます。

任意の引数であるrefArrayを使用すると、リスト中で複数の項目を同時に選択することができます。refArrayには必ず配列を渡し、その各要素は選択する項目の固定参照番号を示します。
その場合、結果として選択された各項目の中で、itemRef 引数に指定した項目がそのリストの新しいカレント項目になります。この項目は、配列によって定義した一連の項目に含まれていない可能性もあります。カレント項目とは、具体的に言うと、EDIT ITEMコマンドを使用した場合に編集モードに移行される項目のことです。

Note: 階層リスト内で複数のリスト項目を同時に選択するには（手動、あるいはプログラムから）、そのリストに対して複数選択可プロパティを有効に設定しておかなくてはなりません。このプロパティの設定は、SET LIST PROPERTIESコマンドを使用して行います。

項目参照番号を使用して作業を実行する場合、項目がユニークな参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細についてはAPPEND TO LISTの説明を参照してください。

例題

hListは一意の参照番号が付いた項目を持つリストです。以下のボタン用オブジェクトメソッドは、現在選択されている項目に親項目が存在する場合にはその親項目を選択します:

```
$vlItemPos:=Selected list items(hList) ` 選択された項目の位置を取得
GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText) ` 選択された項目の参照番号を取得
$vlParentItemRef:=List item parent(hList;$vlItemRef) ` 親項目の参照を取得 （あれば）
If($vlParentItemRef>0)
   SELECT LIST ITEM BY REFERENCE(hList;List item parent(hList;$vlItemRef)) ` 親項目を選択
End if
```

Selected list items

Selected list items ({* ;} list {; itemsArray {; *}}) -> 戻り値

引数	型		説明
*	演算子	→	指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→	リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
itemsArray	倍長整数配列	←	2番目の * 省略時: 配列にはリスト中で選択された 項目の位置配列が返される 2番目の * 指定時: 配列には選択された項目の 参照が返される
*	演算子	→	省略時: 項目位置 指定時: 項目参照
戻り値	倍長整数	↩	2番目の * 省略時: 展開/折りたたまれたされたリスト中 現在選択されている項目位置 2番目の * 指定時: 選択されている項目の参照

説明

Selected list items コマンドは、list引数に渡された参照番号またはオブジェクト名のリストにおいて、選択された項目の位置または参照番号を返します。

オプションの第一引数 * を渡すと、list 引数はフォーム上のリストオブジェクトに対応するオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数は階層リスト参照 (ListRef) です。リストオブジェクトを一つしか使わない場合または2番目の * を渡して項目参照番号を扱う場合、両方のシンタックスを使用できます。他方フォーム上に同じ階層リストを参照する複数のオブジェクトがあり、2番目の * を省略して項目位置を扱う場合、それぞれのオブジェクトが個別に展開/折りたたみ状態をもつので、オブジェクト名に基づくシンタックスを使用しなければなりません。

Note: オブジェクト名に @ 文字を使用することで、名前に対応するオブジェクトが複数検索された場合、Selected list items コマンドは最初に見つけたオブジェクトを処理の対象とします。

複数の項目が選択される場合、コマンドはitemsArray 配列に選択された項目の参照または位置を返します。ユーザが選択した項目を知るために、フォーム上のリストに対しこのコマンドを適用できます。

2番目の * 引数は、項目位置を扱うか (省略時)、項目参照を扱うか (指定時) を指定します。

itemsArray引数には倍長整数タイプの配列を渡すことができます。必要に応じ、関数は配列の作成やサイズ調整を行います。コマンドの実行後、itemsArrayには次の要素が代入されます:

- 2番目の引数 * を省略した場合、リストの展開/折りたたみ状態に相対的な各選択項目の位置。
- 2番目の引数 * を渡した場合、各選択項目の固定参照。

選択された項目が存在しない場合、空の配列が返されます。

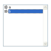
Note: 選択項目が複数存在する場合、コマンドはlistのカレント項目位置あるいは参照番号を戻り値として返します。カレント項目は、ユーザが最後にクリックした項目（手動による選択の場合）またはSELECT LIST ITEMS BY POSITIONやSELECT LIST ITEMS BY REFERENCEコマンドで最後に指定された項目（プログラムによる選択の場合）のいずれかになります。

リストにサブリストがある場合、このコマンドサブリストではなく、メインリスト（フォームで実際に定義されたリスト）に適用します。位置は、リストとそのサブリストの現在の展開または縮小状態を用いて、メインリストの最上位の項目と相対的に表わされます。

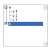
いずれの場合でも、選択された項目が存在しない場合、関数は0を返します。

例題

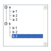
次は、アプリケーションモードで表示されたhListという名前のリストです:



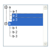
```
$vlItemPos:=Selected list items(hList) ` この時点で$vlItemPos は 2
```



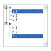
```
$vlItemPos:=Selected list items(hList) ` この時点で$vlItemPos は4
$vlItemRef:=Selected list items(hList;*) ` $vlItemRef は200 (例えば)
```



```
$vlItemPos:=Selected list items(hList) ` この時点で$vlItemPos は8
$vlItemRef:=Selected list items(hList;*) ` $vlItemRef は203 (例えば)
```



```
$vlItemPos:=Selected list items(hList;$arrPos) ` この時点で$vlItemPos は3
 ` $arrPos{1} は3, $arrPos{2} は 4 そして $arrPos{3} は 5
```



```
$vlItemRef:=Selected list items(hList;$arrRefs;*) ` $vlItemRef は203 (例えば) ` $arrRefs{1} は101, $arrRefs{2} は203
(例えば)
```

SET LIST ITEM

SET LIST ITEM ({* ;} list ; itemRef | * ; newItemText ; newItemRef {; sublist ; expanded})

引数	型		説明
*	演算子	→	指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→	リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
itemRef | *	演算子, 倍長整数	→	項目参照番号, または 0: リストに最後に追加された項目, または *: リスト中のカレント項目
newItemText	文字	→	新しい項目テキスト
newItemRef	倍長整数	→	新しい項目参照番号
sublist	ListRef	→	項目に添付する新しいサブリスト, または 0: サブリストがない場合 (現在のサブリストを取り除く), または -1: 変更しない
expanded	ブール	→	オプションのサブリストの展開/折りたたみ

説明

SET LIST ITEM コマンドは、list引数に渡した参照番号またはオブジェクト名のリストにおいて、itemRefで指定した項目を変更します。

1番目の * 引数を渡した場合、list 引数はフォーム中のリストオブジェクトのオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数はリスト参照番号 (ListRef) です。1つしかフォーム上でリストオブジェクトを使用しない場合、いずれのシンタックスも使用できます。同じリストの複数のリストオブジェクトがフォーム上にあり、2番目の * を渡してカレントの項目を処理する場合、リストオブジェクトごとにカレント項目が異なるため、オブジェクト名に基づくシンタックスを使用しなければなりません。

itemRefには参照番号を渡すことができます。渡した項目参照番号を持つ項目が存在しない場合、コマンドは何も行いません。オプションとしてitemRefに0を渡し、APPEND TO LISTコマンドを用いてリストへ最後に追加される項目を指定することができます。
最後に、itemRefには * を渡すことができます。この場合、コマンドはリストのカレント項目に対して適用されます。手動で複数のリスト項目が選択されている場合、最後に選択された項目がカレントリスト項目になります。選択された項目が存在しない場合、コマンドは何も行いません。

項目参照番号を使用して作業を実行する場合、項目がユニークな参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については の説明を参照してください。

newItemTextに項目の新しいテキストを渡します。項目参照番号を変更する場合、newItemRefに新しい値を渡します。変更しない場合は同じ値を渡します。

項目にサブリストを添付する場合、サブリストの参照番号をsubListに渡します。この時、新たなサブリスを展開するにはexpandedにTrueを、そうでない場合はFalseを渡します。

項目にすでに添付されているサブリストを切り離す場合は、sublistに0を渡します。この場合、GET LIST ITEMコマンドを使用して、そのリストの参照番号をあらかじめ取得しておくとよいでしょう。そうすれば、リストが必要なくなったときにCLEAR LISTコマンドを使用して削除することができます。

項目のサブリストプロパティを変更しない場合は、sublistに-1を渡します。

Note: 引数sublistとexpandedはともにオプションですが、指定する場合は組み合わせて指定してください。

例題 1

hListはユニークな参照番号が付いた項目を持つリストです。以下のボタン用オブジェクトメソッドは、現在選択されているリスト項目にサブ項目を追加します。

```
$vlItemPos:=Selected list items(hList)
If($vlItemPos>0)
   GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText;$hSublist;$vbExpanded)
   $vbNewSubList:=Not(Is a list($hSublist))
   If($vbNewSubList)
      $hSublist:=New list
   End if
   vlUniqueRef:=vlUniqueRef+1
   APPEND TO LIST($hSubList;"New Item";vlUniqueRef)
   If($vbNewSubList)
      SET LIST ITEM(hList;$vlItemRef;$vsItemText;$vlItemRef;$hSublist;True)
   End if
   SELECT LIST ITEMS BY REFERENCE(hList;vlUniqueRef)
End if
```

例題 2

GET LIST ITEMコマンドの例題参照

例題 3

APPEND TO LISTコマンドの例題参照

SET LIST ITEM FONT

SET LIST ITEM FONT ({* ;} list ; itemRef | * ; font)

引数	型		説明
*	演算子	→	指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→	リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
itemRef | *	倍長整数, 演算子	→	項目参照番号, または 0: リストに最後に追加された項目, または *: リスト中のカレント項目
font	文字, 倍長整数	→	フォント名または番号

説明

SET LIST ITEM FONT コマンドは、リスト参照またはオブジェクト名をlistに指定したリスト中、itemRef引数で指定した項目の文字フォントを設定します。

1番目の * 引数を渡した場合、list 引数はフォーム中のリストオブジェクトのオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数はリスト参照番号 (ListRef) です。1つしかフォーム上でリストオブジェクトを使用しない、または2番目の * を省略してリスト構造を処理対象とする場合、いずれのシンタックスも使用できます。同じリストの複数のリストオブジェクトがフォーム上にあり、2番目の * を渡してカレントの項目を処理する場合、リストオブジェクトごとにカレント項目が異なるため、オブジェクト名に基づくシンタックスを使用しなければなりません。

itemRefには参照番号を渡すことができます。渡した項目参照番号を持つ項目が存在しない場合、コマンドは何も行いません。オプションとしてitemRefに0を渡し、APPEND TO LISTコマンドを用いてリストへ最後に追加される項目を指定することができます。

最後に、itemRefには * を渡すことができます。この場合、コマンドはリストのカレント項目に対して適用されます。手動で複数のリスト項目が選択されている場合、最後に選択された項目がカレントリスト項目になります。選択された項目が存在しない場合、コマンドは何も行いません。

font 引数には、使用するフォントの名前または番号を渡します。デフォルトフォントを設定するには空の文字列を渡します。

例題

リストのカレント項目にTimesフォントを適用します:

```
 SET LIST ITEM FONT(*;"Mylist";*;"Times")
```

SET LIST ITEM ICON

SET LIST ITEM ICON ({* ;} list ; itemRef | * ; icon)

引数	型		説明
*	演算子	→	指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→	リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
itemRef | *	倍長整数, 演算子	→	項目参照番号, または 0: リストに最後に追加された項目, または *: リスト中のカレント項目
icon	ピクチャー	→	項目に割り当てるアイコン

説明

SET LIST ITEM ICON コマンドは、リスト参照またはオブジェクト名をlistに指定したリスト中、itemRef引数で指定した項目に割り当てるアイコンを設定します。

Note: SET LIST ITEM PROPERTIES コマンドを使用して、項目に割り当てるアイコンを設定することができます。しかしSET LIST ITEM PROPERTIESはスタティックピクチャ参照 (リソース参照またはピクチャライブラリのピクチャ) のみを受け入れます。

1番目の * 引数を渡した場合、list 引数はフォーム中のリストオブジェクトのオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数はリスト参照番号 (ListRef) です。1つしかフォーム上でリストオブジェクトを使用しない、または2番目の * を省略してリスト構造を処理対象とする場合、いずれのシンタックスも使用できます。同じリストの複数のリストオブジェクトがフォーム上にあり、2番目の * を渡してカレントの項目を処理する場合、リストオブジェクトごとにカレント項目が異なるため、オブジェクト名に基づくシンタックスを使用しなければなりません。

itemRefには参照番号を渡すことができます。渡した項目参照番号を持つ項目が存在しない場合、コマンドは何も行いません。オプションとしてitemRefに0を渡し、APPEND TO LISTコマンドを用いてリストへ最後に追加される項目を指定することができます。
最後に、itemRefには * を渡すことができます。この場合、コマンドはリストのカレント項目に対して適用されます。手動で複数のリスト項目が選択されている場合、最後に選択された項目がカレントリスト項目になります。選択された項目が存在しない場合、コマンドは何も行いません。

icon 引数には有効な4Dピクチャ式 (フィールド、変数、ポインタなど) を渡します。ピクチャは項目の左側に表示されます。
階層リストを最適化されるため、ポインタの利用を特に推奨します。異なるリスト項目に複数回同じアイコンが使用される場合でも、ポインタを使用すればメモリ上に一つだけピクチャインスタンスが作成されます。

Note: 逆にGET ICON RESOURCE や GET PICTURE RESOURCE コマンドで作成されたピクチャを直接利用することは推奨されません。アイコンはリストごとにメモリ上に複製されます。

例題

このコードはポインタを利用しているため最適化されています:

```
C_POINTER(vIcon)
vIcon:=->[Params]Icon
SET LIST ITEM ICON(mylist;ref1;vIcon->)
SET LIST ITEM ICON(mylist;ref2;vIcon->)
```

SET LIST ITEM PARAMETER

SET LIST ITEM PARAMETER ({* ;} list ; itemRef | * ; selector ; value)

引数	型	説明
*	演算子	⇒指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	⇒リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
itemRef | *	演算子, 倍長整数	⇒項目参照番号, または 0: リストに最後に追加された項目, または *: リスト中のカレント項目
selector	文字	⇒パラメタ定数
value	文字, ブール, 倍長整数, 実数	⇒パラメタ値

説明

SET LIST ITEM PARAMETERコマンドは、リスト参照またはオブジェクト名をlistに指定したリスト中、itemRef引数で指定した項目のselector パラメタを設定するために使用します。

1番目の * 引数を渡した場合、list 引数はフォーム中のリストオブジェクトのオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数はリスト参照番号 (ListRef) です。1つしかフォーム上でリストオブジェクトを使用しない、または2番目の * を省略してリスト構造を処理対象とする場合、いずれのシンタックスも使用できます。同じリストの複数のリストオブジェクトがフォーム上にあり、2番目の * を渡してカレントの項目を処理する場合、リストオブジェクトごとにカレント項目が異なるため、オブジェクト名に基づくシンタックスを使用しなければなりません。

itemRefには参照番号を渡すことができます。渡した項目参照番号を持つ項目が存在しない場合、コマンドは何も行いません。オプションとしてitemRefに0を渡し、APPEND TO LISTコマンドを用いてリストへ最後に追加される項目を指定することができます。
最後に、itemRefには * を渡すことができます。この場合、コマンドはリストのカレント項目に対して適用されます。手動で複数のリスト項目が選択されている場合、最後に選択された項目がカレントリスト項目になります。選択された項目が存在しない場合、コマンドは何も行いません。

selectorにはHierarchical ListsテーマのAdditional text定数または任意のカスタム値を渡すことができます:

- Additional Text: この定数はitemRef項目の右側にテキストを追加するために使用します。この追加されたタイトルは、ユーザが水平スクロールバーを使用した場合でも、リストの右側に常に表示されます。この定数を使用する場合、表示するテキストをvalueに渡します。
- カスタムセレクタ: カスタムテキストセレクタと、対応するテキスト、数値、ブール型の値を指定できます。この値はリスト項目に格納され、GET LIST ITEM PARAMETERコマンドを使用して取り出すことができます。これにより階層リストに関連づけられたインタフェースをセットアップできます。例えば顧客名リストにおいて、年齢をリスト項目に関連付け、項目が選択されたときにそれを表示させることができます。

SET LIST ITEM PROPERTIES

SET LIST ITEM PROPERTIES ({* ;} list ; itemRef | * ; enterable ; styles ; icon {; color})

引数	型	説明
*	演算子	→指定時, listはオブジェクト名 (文字列) 省略時, listはリスト参照番号
list	ListRef, 文字	→リスト参照番号 (* 省略時), または リストオブジェクト名 (* 指定時)
itemRef | *	演算子, 倍長整数	→項目参照番号, または 0: リストに最後に追加された項目, または *: リスト中のカレント項目
enterable	ブール	→TRUE = 入力可, FALSE = 入力不可
styles	倍長整数	→項目のフォントスタイル
icon	倍長整数	→"cicn" Mac OSベースのリソースID, または 65536 + "PICT" Mac OSベースのリソースID, または 131072 + ピクチャ参照番号
color	倍長整数	→RGBカラー値 または -1 = 元のカラーにリセット

説明

SET LIST ITEM PROPERTIES コマンドは、引数listに渡された参照番号またはオブジェクト名のリスト内で、itemRefによって指定された項目を変更します。

オプションの第一引数 * を渡すと、list 引数はフォーム上のリストオブジェクトに対応するオブジェクト名 (文字列) です。この引数を渡さない場合、list 引数は階層リスト参照 (ListRef) です。リストオブジェクトを一つしか使わない場合や、2番目の * を使用しない場合は、両方のシンタックスを使用できます。他方フォーム上に同じ階層リストを参照する複数のオブジェクトがある場合で、2番目の * を渡してカレント項目を参照する場合、それぞれのオブジェクトが個別にカレント項目をもつので、オブジェクト名に基づくシンタックスを使用しなければなりません。

itemRefには参照番号を渡すことができます。渡された項目参照番号を持つ項目が存在しない場合、コマンドは何も動作しません。オプションとしてitemRefに0を渡すことによって、APPEND TO LISTコマンドを使用してリストに追加した最後の項目を変更することができます。

最後に、引数itemRefに * を渡すことができます。この場合、コマンドはリストのカレント項目に対して適用されます。手動で複数のリスト項目が選択されている場合、最後に選択された項目がカレントリスト項目となります。選択された項目が存在しない場合、コマンドは何も行いません。

項目参照番号を使用して作業を実行する場合、項目がユニークな参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、を参照してください。

Note: 項目のテキストまたはそのサブリストを変更するには、SET LIST ITEMコマンドを使用します。

項目を入力可能にする場合は、enterable引数にTRUEを渡し、そうでない場合はFALSEを渡します。

重要: 入力可にするためには、その項目が入力可なリストに属する必要があります。
OBJECT SET ENTERABLEコマンドを使用すると、リスト全体を入力可または入力不可にすることができます。SET LIST ITEM PROPERTIESコマンドを使用すると、個々のリスト項目を入力可または入力不可にすることができます。入力可プロパティをリストレベルで変更しても、項目の入力可プロパティは変更されません。しかし、項目に入力できるのは、そのリストが入力可な場合のみです。

項目のフォントスタイルは、styles引数で指定します。以下の定義済定数の1つ、または複数を組み合わせて渡します:

定数	型	値
Bold	倍長整数	1
Italic	倍長整数	2
Plain	倍長整数	0
Underline	倍長整数	4

項目にアイコンを関連付ける場合、以下の数値のいずれか1つを渡します:

- N: NはMac OSベースの'cicn'リソースのリソースIDです。
- Use PICT resource+Nを使用: NはMac OSベースの'PICT'リソースのリソースIDです。
- Use PicRef+Nを使用: Nはデザインモードのピクチャライブラリのピクチャ参照番号です。

項目にグラフィックを使用しない場合は、ゼロを渡します。

Notes:

- Use PICT resourceとUse PicRefはテーマの定義済み定数です。
- 4D ピクチャ式 (フィールド, 変数, 等) を使用して項目アイコンを指定する場合、SET LIST ITEM ICON コマンドを使用します。

color 引数 (オプション) を使用して、項目テキストの色を変更することができます。この色はRGBフォーマット、例えば0x00RRGGBB形式の4バイトの倍長整数で指定しなくてはなりません。このフォーマットに関する詳細は、OBJECT SET RGB COLORSコマンドの説明を参照してください。項目をオリジナルの色にリセットするには、color引数に-1を渡します。

例題 1

APPEND TO LISTコマンドの例題参照

例題 2

以下の例題はカレント項目のテキストを太字の赤に変更します:

```
SET LIST ITEM PROPERTIES(list;*;True;Bold;0;0x00FF0000)
```

SET LIST PROPERTIES

SET LIST PROPERTIES (list ; appearance {; icon {; lineHeight {; doubleClick {; multiSelections {; editable}}}}})

引数	型		説明
list	ListRef	→	リスト参照番号
appearance	倍長整数	→	リストの描画スタイル 1 = Macスタイル 2 = Windowsスタイル 0 = プラットフォームに基づく自動アピアランス
icon	倍長整数	→	"cicn" Mac OSベースのリソースID または 0 デフォルトのプラットフォームノードアイコン
lineHeight	倍長整数	→	最小行高さ (ピクセル単位)
doubleClick	倍長整数	→	ダブルクリックでサブリストを展開/折り畳み 0 = Yes, 1= No
multiSelections	倍長整数	→	複数行選択: 0 = No (デフォルト), 1 = Yes
editable	倍長整数	→	0 = ユーザによるリスト編集不可, 1 = ユーザによるリスト編集可 (デフォルト)

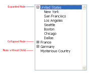
説明

SET LIST PROPERTIES コマンドは、list引数に渡された参照番号を持つ階層リストの表示様式を設定します。

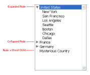
appearance 引数にはHierarchical Listsテーマの定義済定数のいずれかを指定します:

定数	型	値
Ala Macintosh	倍長整数	1
Ala Windows	倍長整数	2

Windowsアピアランスではアイコン (+) は折りたたまれたノード、アイコン (-) は展開されたノードを示します。子項目がないノードにはアイコンが表示されません。以下は、Windowsアピアランスのデフォルトの階層リストです:


Expanded Node
United States
New York
San Francisco
Los Angeles
Seattle
Boston
Chicago
Dallas
Collapsed Node
France
Germany
Node without Child
Mysterious Country

Macintoshアピアランスでは、横向きの三角アイコンは折りたたまれたノード、下向きの三角アイコンは展開されたノードを示します。サブ項目を持たないノードにはアイコンがありません。以下はMacintoshアピアランスのデフォルトの階層リストです:


Expanded Node
United States
New York
San Francisco
Los Angeles
Seattle
Boston
Chicago
Dallas
Collapsed Node
France
Germany
Node without Child
Mysterious Country

SET LIST PROPERTIESコマンドを呼び出さずに階層リストオブジェクトを表示するか、またはappearence引数に0を渡すと、リストはデザイン」モードのフォームエディタでそのオブジェクトに対して選択したプラットフォームインタフェースプロパティに応じて、デフォルトのWindowsまたはMacintoshの様式で表示されます。

引数iconは、それぞれのノードを表示するアイコンを示します。iconに渡された値は折りたたまれたノードのアイコンを設定し、icon+1は展開されたノードのアイコンを設定します。

例えば、15000を渡した場合、折りたたまれた各ノードに対してカラーアイコン'cicn' ID=15000が表示され、展開された各ノードに対してはカラーアイコン'cicn' ID=15001が表示されます。

したがって、これらの'cicn'カラーアイコンリソースがデータベースのストラクチャファイルに存在することが重要です。カラーアイコンがないと、対応するノードはアイコンなしで表示されます(リストをアイコンなしで表示するには、この方法を利用します)。

警告: 'cicn'カラーアイコンリソースを作成する場合、15000以上のリソースIDを使用してください。15000未満のリソースIDは4Dのために予約されています。

デフォルトのMacintoshおよびWindowsノードのリソースIDは、4Dの以下の定義済定数によって表されます:

定数	型	値
Macintosh node	倍長整数	860
Windows node	倍長整数	138

つまり、4Dは以下の'cicn'リソースを提供します:

ID Number	Description
860	Collapsed node a la Macintosh
861	Expanded node a la Macintosh
138	Collapsed node a la Windows
139	Expanded node a la Windows

icon引数を渡さない場合や0を渡した場合、ノードは選択した表示様式のデフォルトのアイコンで表示されます。

カラーアイコンリソースにはさまざまなサイズがあります。例えば16x16や32x32のカラーアイコンを作成することができます。

lineHeight引数を渡さない場合、階層リストの行の高さは、オブジェクトに使用されるフォントおよびフォントサイズによって決定されます。縦または横に大き過ぎるカラーアイコンを使用すると、端が切れて表示されるか、接続のための点線(Windows表示の場合)やその上下のテキストによって消されるか、またはその両方になります。

カラーアイコンのサイズ、フォント、フォントサイズを適切に選択するか、そうでなければlineHeightで階層リストの行の高さの最小値を渡します。渡した値が、使用するフォントやフォントサイズから求められる行の高さより大きい場合、階層リストの行の高さは渡された値になります。

Note: SET LIST PROPERTIESコマンドは、階層リストの中でのノードの表示方法に影響します。リストの中の個々の項目のアイコンをカスタマイズするには、SET LIST ITEM PROPERTIESコマンドを使用します。

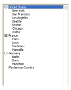
オプションのdoubleClick引数により、親リスト項目をダブルクリックしてサブリストの展開/折りたたみを行うか、行わないかを定義できます。デフォルトでは親リスト項目をダブルクリックすることにより、子リストが開いたり閉じたりするようになっています。しかし、いくつかのユーザーインタフェイスではこの動作が起こらないようにする必要があります。そのためには、doubleClick引数を1に設定します。
ダブルクリックの動作だけが発生しなくなります。リストのノードをクリックすることによりサブリストの開閉ができます。
doubleClick 引数を省略するか0を渡すと、デフォルトの動作が適用されます。

オプションのmultiSelections引数を使用し、リストが複数項目の選択を受け入れるかどうかを指示することができます。
デフォルトでは以前のバージョンの4Dと同様、階層リストの項目を複数同時に選択することはできません。リストに対してこの機能を利用可能にしたい場合は、multiSelections引数に1を渡します。その場合、次の方法で複数選択機能を使用することができます:
- 手動の場合、連続した項目の選択には**Shift+クリック**、連続していない項目の選択には**Ctrl+クリック**(Windows)または**Command+クリック**(Mac OS)を使用します。
- プログラムを使用する場合、SELECT LIST ITEMS BY POSITIONおよびSELECT LIST ITEMS BY REFERENCEコマンドを使用します。
multiSelections引数に0を渡すか省略した場合、デフォルトの動作が適用されます。

オプションの引数editableを使用すると、データ入力中にリストがフィールドや変数に関連付けられた選択リストとして表示された際に、ユーザによる編集を可能にするかどうかを指示することができます。リストが編集可である場合、選択リストウインドウに**変更**ボタンが追加され、ユーザはエディタを用いて、値の追加や削除、並び替えを行うことができます。
editable引数に値1を渡すか省略すると、そのリストは編集可能になります。0を渡すと、リストを編集することはできません。

例題

以下の階層リストが、デザインモードのリストエディタで定義されています:

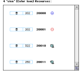

United States
New York
San Francisco
Los Angeles
Seattle
Boston
Chicago
Dallas
France
Paris
Lyon
Bordeaux
Marseille
Germany
Berlin
Bonn
Munchen
Mysterious Country

フォーム中で、hlCities階層リストオブジェクトがこのリストをこのオブジェクトメソッドで再利用します:

```
Case of
   :(Form event=On Load)
      hlCities:=Load list("Cities")
      SET LIST PROPERTIES(hlCities;vlAppearance;vlIcon)
   :(Form event=On Unload)
      CLEAR LIST(hlCities;*)
End case
```

加えて、データベースのストラクチャファイルが以下の'cicn'カラーアイコンリソースを含むように編集されています:

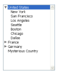
4 'cicn' (Color Icon) Resources:


202
20000
202
20001
322
20010
250
20011

1) 以下の行により:

```
SET LIST PROPERTIES(hlCities;Ala Macintosh;Macintosh node)
```

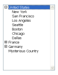
階層リストは、以下のように表示されます:


United States
New York
San Francisco
Los Angeles
Seattle
Boston
Chicago
Dallas
France
Germany
Mysterious Country

2) 以下の行により:

```
SET LIST PROPERTIES(hlCities;Ala Windows;Windows node)
```

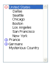
階層リストは、以下のように表示されます:


United States
New York
San Francisco
Los Angeles
Seattle
Boston
Chicago
Dallas
France
Germany
Mysterious Country

3) 以下の行により:

```
SET LIST PROPERTIES(hlCities;Ala Windows;20000)
```

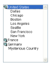
階層リストは、以下のように表示されます:


United States
Dallas
Seattle
Chicago
Boston
Los Angeles
San Francisco
New York
France
Germany
Mysterious Country

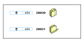
4) 以下の行により:

```
SET LIST PROPERTIES(hlCities;Ala Macintosh;20010)
```

階層リストは、以下のように表示されます:


United States
Dallas
Chicago
Boston
Los Angeles
Seattle
San Francisco
New York
France
Germany
Mysterious Country

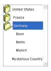
次に、以下に示す'cicn'カラーアイコンリソースをデータベースのストラクチャファイルに追加します:


634
20020
634
20021

5) 以下の行により:

```
SET LIST PROPERTIES(hlCities;Ala Windows;20020;32)
```

階層リストは、以下のように表示されます:


United States
France
Germany
Bonn
Berlin
Munich
Mysterious Country

SORT LIST

SORT LIST (list {; > または <})

引数	型		説明
list	ListRef	⇒	リスト参照番号
> または <	演算子	⇒	並び順: > 昇順, または < 降順

説明

SORT LIST コマンドは、list引数に渡された参照番号を持つリストをソートします。

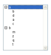
昇順にソートするには>を渡し、降順にソートするには<を渡します。ソート順パラメータを省略した場合、SORT LISTコマンドはデフォルトで昇順にソートします。

SORT LISTコマンドは、すべてのレベルのリストをソートします。まずリストの項目をソートし、次にサブリストがあればそれらをソートし、サブリストの中の項目をソートするというようにリストのすべてのレベルに降りていきます。通常、SORT LISTコマンドをフォームに表示されているリストに適用するのはこのためです。サブリストのソートは、上位レベルを呼び出したときに順序が変更されるので、ほとんど意味がありません。

SORT LIST コマンドは、カレントリスト項目またはリストやサブリストの現在の展開または縮小状態を変更しません。ただし、カレント項目が並び替えによって移動されることがあるため、Selected list itemsは並び替えの前と後で異なる位置を返す可能性があります。

例題

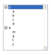
リスト名をhListとし、アプリケーションモードで表示します（Windows形式）:



以下のコードを実行します:

```
` リストを昇順にソート
SORT LIST(hList;>)
```

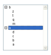
リストは以下のようになります:



このコードを実行すると:

```
` リストを降順にソート
SORT LIST(hList;<)
```

このようになります:



REDRAW LIST

REDRAW LIST (list)

引数型		説明
list	ListRef	→ リスト参照番号

説明

互換性メモ: 4D v11より、REDRAW LIST コマンドは意味を持たなくなりました。階層リストオブジェクトは常に自動で再描画されます。このコマンドを呼び出しても、コマンドは何も行いません。

定数テーマリスト

- 4D Environment
- ASCII Codes
- Backup and Restore
- BLOB
- Colors
- Communications
- Data File Maintenance
- Database Engine
- Database Events
- Database Parameters
- Date Display Formats
- Days and Months
- Design Object Access
- Dictionaries
- Digest Type
- Euro Currencies
- Events (Modifiers)
- Events (What)
- Expressions
- Field and Variable Types
- Find Window
- Font Styles
- Font Type List
- Form Area
- Form Events
- Form Object Types
- Form Objects (Access)
- Form Objects (Properties)
- Form Parameters
- Function Keys
- Hierarchical Lists
- HTTP Client
- Index Type
- Is License Available
- ISO Latin Character Entities
- List Box
- Listbox Footer Calculation
- Log Events
- Math
- Menu Item Properties
- Multistyle Text
- Multistyle Text Attributes
- Open Form Window
- Open window
- Pasteboard
- PHP
- Picture Compression
- Picture Display Formats
- Picture Metadata Names
- Picture Metadata Values
- Picture Transformation
- Platform Interface
- Platform Properties
- Print Options
- Process State
- Process Type
- QR Area Properties
- QR Borders
- QR Commands
- QR Document Properties
- QR Operators
- QR Output Destination
- QR Report Types
- QR Rows for Properties
- QR Text Properties
- Queries
- Relations
- Resources Properties
- SCREEN DEPTH
- SET RGB COLORS
- Shortcut and Associated Keys
- SQL
- Standard System Signatures
- System Documents
- System Folder
- System Format
- TCP Port Numbers
- Time Display Formats
- Trigger Events
- Value for Associated Standard Action
- Web Area
- Web Server
- Web Services (Client)
- Web Services (Server)
- Window Kind
- XML
- 関連付けられた標準アクションのテキスト値

4D Environment

定数	型	値	コメント
_O_4D Desktop	倍長整数	3	
_O_4D Interpreted Desktop	倍長整数	2	
_O_Full Version	倍長整数	0	
4D Client Database Folder	倍長整数	3	
4D Local Mode	倍長整数	0	
4D Remote Mode	倍長整数	4	
4D Server	倍長整数	5	
4D Volume Desktop	倍長整数	1	
64 bit Version	倍長整数	1	
Active 4D Folder	倍長整数	0	
Current localization	倍長整数	1	アプリケーションのカレント言語: デフォルト言語またはSET DATABASE LOCALIZATIONコマンドで設定された言語。
Current Resources Folder	倍長整数	6	
Database Folder	倍長整数	4	
Database Folder Unix Syntax	倍長整数	5	
Default localization	倍長整数	0	Resourcesフォルダとシステム環境に基づき、4Dが起動時に自動で設定する言語 (変更不可)。
Demo Version	倍長整数	0	
Extras Folder	倍長整数	2	
Full method text	倍長整数	1	
Highlighted method text	倍長整数	2	
HTML Root Folder	倍長整数	8	
Internal 4D localization	倍長整数	3	並び替えやテキスト比較で4Dが使用する言語 (アプリケーションの環境設定で設定)。
Licenses Folder	倍長整数	1	
Logs Folder	倍長整数	7	
Merged application	倍長整数	2	バージョンは 4D Volume Desktopを使用して組み込まれたアプリケーションです。
Structure Settings	倍長整数	0	ストラクチャー設定を使用 (引数が省略された際のデフォルト)。このモードでは使用されるselectorの値は標準モードと同じです。
User Settings	倍長整数	1	ユーザー設定を使用。このモードでは特定のキーのみをselector引数で使用できます。
User system localization	倍長整数	2	システムのカレントユーザーが設定した言語

ASCII Codes

定数	型	値	コメント
ACK ASCII code	倍長整数	6	
At sign	倍長整数	64	
Backspace	倍長整数	8	
BEL ASCII code	倍長整数	7	
BS ASCII code	倍長整数	8	
CAN ASCII code	倍長整数	24	
Carriage return	倍長整数	13	
CR ASCII code	倍長整数	13	
DC1 ASCII code	倍長整数	17	
DC2 ASCII code	倍長整数	18	
DC3 ASCII code	倍長整数	19	
DC4 ASCII code	倍長整数	20	
DEL ASCII code	倍長整数	127	
DLE ASCII code	倍長整数	16	
Double quote	倍長整数	34	
EM ASCII code	倍長整数	25	
ENQ ASCII code	倍長整数	5	
Enter	倍長整数	3	
EOT ASCII code	倍長整数	4	
ESC ASCII code	倍長整数	27	
Escape	倍長整数	27	
ETB ASCII code	倍長整数	23	
ETX ASCII code	倍長整数	3	
FF ASCII code	倍長整数	12	
FS ASCII code	倍長整数	28	
GS ASCII code	倍長整数	29	
HT ASCII code	倍長整数	9	
LF ASCII code	倍長整数	10	
Line feed	倍長整数	10	
NAK ASCII code	倍長整数	21	
NBSP	倍長整数	202	
NUL ASCII code	倍長整数	0	
Period	倍長整数	46	
Quote	倍長整数	39	
RS ASCII code	倍長整数	30	
SI ASCII code	倍長整数	15	
SO ASCII code	倍長整数	14	
SOH ASCII code	倍長整数	1	
SP ASCII code	倍長整数	32	
Space	倍長整数	32	
STX ASCII code	倍長整数	2	
SUB ASCII code	倍長整数	26	
SYN ASCII code	倍長整数	22	
Tab	倍長整数	9	
US ASCII code	倍長整数	31	
VT ASCII code	倍長整数	11	

Backup and Restore

定数	型	値	コメント
Last Backup Date	倍長整数	0	
Last Backup Status	倍長整数	2	
Last Restore Date	倍長整数	0	
Last Restore Status	倍長整数	2	
Next Backup Date	倍長整数	4	

BLOB

定数	型	値	コメント
Compact compression mode	倍長整数	1	圧縮解凍の処理速度と引き換えに、BLOBをできるだけ小さく圧縮します。デフォルトモード。
Extended real format	倍長整数	1	
Fast compression mode	倍長整数	2	圧縮率と引き換えにBLOBをできるだけ速く圧縮・解凍します (圧縮されたBLOBのサイズは大きくなります)。
GZIP Best compression mode	倍長整数	-1	GZIP圧縮で圧縮率を優先します。
GZIP Fast compression mode	倍長整数	-2	GZIP圧縮で速度を優先します。
Is not compressed	倍長整数	0	
Mac C string	倍長整数	0	
Mac Pascal string	倍長整数	1	
Mac Text with length	倍長整数	2	
Mac Text without length	倍長整数	3	
Macintosh byte ordering	倍長整数	1	
Macintosh double real format	倍長整数	2	
Native byte ordering	倍長整数	0	
Native real format	倍長整数	0	
PC byte ordering	倍長整数	2	
PC double real format	倍長整数	3	
UTF8 C string	倍長整数	4	
UTF8 Text with length	倍長整数	5	
UTF8 Text without length	倍長整数	6	

Colors

定数	型	値	コメント
Black	倍長整数	15	
Blue	倍長整数	6	
Brown	倍長整数	13	
Dark Blue	倍長整数	5	
Dark Brown	倍長整数	10	
Dark Green	倍長整数	9	
Dark Grey	倍長整数	11	
Green	倍長整数	8	
Grey	倍長整数	14	
Light Blue	倍長整数	7	
Light Grey	倍長整数	12	
Orange	倍長整数	2	
Purple	倍長整数	4	
Red	倍長整数	3	
White	倍長整数	0	
Yellow	倍長整数	1	

Communications

定数	型	値	コメント
Data bits 5	倍長整数	0	
Data bits 6	倍長整数	2048	
Data bits 7	倍長整数	1024	
Data bits 8	倍長整数	3072	
MacOS Printer Port	倍長整数	0	
MacOS Serial Port	倍長整数	1	
Parity Even	倍長整数	12288	
Parity None	倍長整数	0	
Parity Odd	倍長整数	4096	
Protocol DTR	倍長整数	30	
Protocol None	倍長整数	0	
Protocol XONXOFF	倍長整数	20	
Speed 115200	倍長整数	1022	
Speed 1200	倍長整数	94	
Speed 1800	倍長整数	62	
Speed 19200	倍長整数	4	
Speed 230400	倍長整数	1021	
Speed 2400	倍長整数	46	
Speed 300	倍長整数	380	
Speed 3600	倍長整数	30	
Speed 4800	倍長整数	22	
Speed 57600	倍長整数	0	
Speed 600	倍長整数	189	
Speed 7200	倍長整数	14	
Speed 9600	倍長整数	10	
Stop bits One	倍長整数	16384	
Stop bits One and a half	倍長整数	-32768	
Stop bits Two	倍長整数	-16384	

Data File Maintenance

定数	型	値	コメント
Compact address table	倍長整数	131072	強制的にレコードのアドレステーブルを更新します (圧縮時間は長くなります)。このオプションのみを指定すると、4Dは自動でレコードの再保存オプションを有効にします。この場合、レコード番号が更新される点に留意してください。
Create process	倍長整数	32768	このオプションが渡されると圧縮は非同期で行われ、コールバックメソッドを使用して結果を管理しなければなりません。4Dは進捗状況を表示しません (コールバックメソッドを使用して表示させることができます)。プロセスが正しく起動されるとOKシステム変数が1に設定され、他の場合は0に設定されます。このオプションが渡されない時、圧縮が行われればOK変数に1が設定され、そうでなければ0が設定されます。
Do not compact index	倍長整数	2	
Do not create log file	倍長整数	16384	通常このコマンドはXMLフォーマットのログファイルを作成します。このオプションを使用すればログファイルは作成されません。
Move to Replaced files folder	倍長整数	4	
New file	倍長整数	0	
New file dialog	倍長整数	1	
Renumber records	倍長整数	1	
Update records	倍長整数	65536	現在のストラクチャー定義に基づき、すべてのレコードを強制的に再保存します。
Use default folder	倍長整数	1	引数として渡されたフィールドのデータは*databaseName*.ExternalDataという名前のデフォルトフォルダーに保存されます。このフォルダーはデータファイルと同階層に作成されます。このモードでは、外部データを、それがデータファイル内にあるときと同様に、4Dが管理します。
Use selected file	倍長整数	2	
Use structure definition	倍長整数	0	4Dはストラクチャーに設定されたフィールドデータの格納設定を使用します。外部ストレージから内部ストレージに変更しても、外部ファイルは削除されません。
Verify All	倍長整数	16	
Verify Indexes	倍長整数	8	このオプションを使用すると、インデックスの物理的な整合性を検証しますが、データとのリンクは考慮されません。この検証は無効なキーを検知しますが、重複キー (同じレコードを参照する2つのインデックス) を検出することはできません。この検証を行うにはVerify Allオプションを使用しなければなりません。
Verify Records	倍長整数	4	

Database Engine

定数	型	値	コメント
New record	倍長整数	-3	
No current record	倍長整数	-1	

Database Events

定数	型	値	コメント
On after host database exit	倍長整数	4	ホストデータベースの On Exitデータベースメソッド データベースメソッドが実行を終了したところです。
On after host database startup	倍長整数	2	ホストデーターベースの On Startupデータベースメソッド データベースメソッドが実行を終了したところです。
On application background move	倍長整数	1	4Dアプリケーションがバックグランドに移動した
On application foreground move	倍長整数	2	4Dアプリケーションが最前面に移動した
On before host database exit	倍長整数	3	ホストデータベースは閉じられようとしているところです。ホストデータベースの On Exitデータベースメソッド データベースメソッドはまだ呼び出されていない状態です。 ホストデータベースの On Exitデータベースメソッド データベースは、コンポーネントの On Host Database Event データベースメソッド データベースメソッドが実行されている間は呼び出されません。
On before host database startup	倍長整数	1	ホストデータベースはちょうど開かれたところです。ホストデータベースの On Startupデータベースメソッド データベースメソッドはまだ呼び出されていません。 On Startupデータベースメソッド データベースメソッドは、On Host Database Event データベースメソッド データベースメソッドがコンポーネント内で実行されている間は呼び出されません。

Database Parameters

定数	型	値	コメント
_O_Real display precision	倍長整数	32	*** このセレクターは廃止されました ***
4D Local Mode Scheduler	倍長整数	10	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: Yes **説明**: セレクタ12参照
4D Remote Mode Scheduler	倍長整数	12	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ10, 11 および 12に対し、value引数は16進数、0x00aabbccの形式で表わされます。詳細は次の通りです: *aa* = システムへのコール毎の最小tick数（0～100） *bb* = システムへのコール毎の最大tick数（0～100） *cc* = システムへのコール間のtick数（0～20） これらの値が範囲外のとき、4Dはその値を最大数に設定します。value引数には、次の定義済標準値のうちいずれかを渡すことができます: • value = -1: 4Dに最高優先度を割り当てる • value = -2: 4Dに平均的な優先度を割り当てる • value = -3: 4Dに最低優先度を割り当てる **説明**: この引数を使用して、4Dシステム内部コールをダイナミックに設定することができます。selectorに応じて、スケジューラの値は次のアプリケーションのために設定されます。 • シングルユーザの4Dから呼び出された場合、ローカルモードの4D (selector=10). • 4D Serverから呼び出された場合、4D Server (selector=11). • 4DServerに接続した4Dから呼び出された場合、リモートモードの4D (selector=12). **Note:** セレクタ=12(4D Remote Mode Scheduler)は、SET DATABASE PARAMETERコマンドがサーバーマシン上で実行されるか、クライアントマシン上で実行されるかにより異なります: • コマンドがサーバーマシン上で実行される場合、新しい値はコマンド実行後にサーバに接続する全てのクライアントマシンに適用されます。 • コマンドがクライアントマシン上で実行される場合、新しい値はそのクライアントマシンで直ちに有効となり、またコマンド実行後にサーバに接続するすべてのクライアントマシンに対し適用されます。 この動作を使用することで、クライアントマシン毎に異なる特性をダイナミックに扱うことが可能です。これはコマンドの実行がまず、そのクライアントマシンの設定に影響し、その後にサーバに接続するすべてのクライアントマシンへの設定値として使用されるという動作に基づきます。 この動作は、4Dの6.8.6, 2003.3と4D2004以降のバージョンで効果があります。 **警告**: これらのセレクタを不適切に使用すると、アプリケーションパフォーマンスが深刻に低下する原因となります。これらの要素に対する完全な知識がある場合のみ変更を行うことをお勧めします。
4D Remote Mode Timeout	倍長整数	14	**スコープ**: value が正数の場合4D アプリケーション **2セッション間で設定を保持**: value が正数の場合Yes **説明**: この引数を使用して、4D Serverに接続したリモートの4Dマシンが参照するタイムアウト値を設定できます。リモートモードの4Dが使用するデフォルトのタイムアウト値は、リモートマシン上の環境設定の"クライアント-サーバ/設定"ページで設定されます。 このセレクタに関する詳細は4D Server Timeout (13) の説明を参照してください。4D Remote Mode Timeout セレクタは非常に特殊な状況において使用されます。
4D Server Log Recording	倍長整数	28	**Scope**: 4D Server, 4D リモート **2セッション間で設定を保持**: No **とりうる値**: 0 または 1 から X (0 = 記録しない, 1 から X = ファイル名に付加するシーケンス値)。 **説明**: 4D Serverが受け取る標準的なリクエスト（Webリクエストを除く）の記録を開始または停止します。デフォルト値は0（リクエストを記録しない）です。 サーバマシンが受信した各リクエストをログファイルに記録するよう、4D Serverに指示することができます。このメカニズムが有効になると、データベースストラクチャと同じ階層にログファイルが作成されます。ファイルには"4DRequestsLogX"（Xはログのシーケンシャル番号）の名前が付けられます。ファイルサイズが10MBに達するとそのファイルは閉じられ、インクリメントしたシーケンス番号を使用して新しいファイルが作成されます。もし同じ名前のファイルが存在する場合は置き換えられます。シーケンシャル番号の開始番号を、value引数を使用して指定できます。 このテキストファイルにはそれぞれのリクエストに関する様々な情報（時間、プロセス番号、ユーザ、リクエストサイズ、処理時間など）がシンプルな表形式のフォーマットで保存されます。この情報はアプリケーションの調整を行う場合や統計の目的で利用する場合に特に役立ちます。この情報を、例えばスプレッドシートソフトウェアに読み込んで処理することもできます。 **Note:** マニュアル操作により、ログの記録をOn/Offすることが出来ます。Windowsでは**Ctrl+Alt+L**、MacOSでは**Command+Option+L**のショートカットです。
4D Server Scheduler	倍長整数	11	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: Yes **説明**: セレクタ12参照
4D Server Timeout	倍長整数	13	**スコープ**: value が正数なら4Dアプリケーション **2セッション間で設定を保持**: value が正数ならYes **とりうる値**: 0 -> 32 767 **説明**: この引数を使用して、4D Serverのタイムアウトの値を変更することができます。4D Serverのタイムアウトのデフォルト値は、サーバ側の「環境設定」ダイアログボックスの「クライアント-サーバ/接続設定」ページで定義します。 セレクタ4D Server Timeoutにより、対応する引数valueの新しいタイムアウト（分単位で指定）を設定できます。この機能は、クライアント側でCPUを占有する、時間がかかる処理を実行する前に、タイムアウト設定を長くしたい場合は特に便利です。例えば、膨大なページの印刷などは、予期しないタイムアウトになる可能性があります。 2種類のオプションがあります: • value引数に**正数**を渡すと、グローバルかつ永続的なタイムアウトが設定されます。この新しい値はすべてのプロセスに対して適用され、4Dアプリケーションの環境設定に保存されます（環境設定ダイアログボックスで変更した場合と同じ）。 • value引数に**負数**を渡すと、ローカルで一時的なタイムアウトが設定されます。この新しい値は呼び出し元のプロセスに対してのみ適用され（他のプロセスではデフォルトの値を維持）、例えば処理の終了時のように、クライアントが動作していることを示す信号をサーバが受信すると即座に、デフォルト値へリセットされます。このオプションは、4Dプラグインにより開始された時間のかかる処理を管理する際に便利です。 "タイムアウトしない"オプションを設定するには、valueに0 を渡します。
Auto Synchro Resources Folder	倍長整数	48	**スコープ**: 4D リモートマシン **2セッション間で設定を保持**: No **とりうる値**: 0 (同期しない), 1 (自動同期) または2 (確認する). **説明**: このコマンドを実行する4DクライアントマシンのResourcesフォルダの動的な同期モード。 サーバ上のResourcesフォルダの内容が更新されたり、(リソースエクスプローラや SET DATABASE LOCALIZATION コマンドで) ユーザが同期をリクエストすると、サーバは接続されたユーザに通知を行います。 クライアント側では3つの同期モードを選択できます。Auto Synchro Resources Folderセレクタはカレントセッションでクライアントマシンが使用するモードを指定できます: • 0 (デフォルト値): 動的な同期を行わない (同期リクエストは無視される) • 1: 自動の動的同期 • 2: クライアントマシンにダイアログを表示し、同期の受け入れ回避を確認する アプリケーションの環境設定で、同期モードをグローバルに設定できます。
Cache unload minimum size	倍長整数	66	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**: 1より大きい正の倍長整数 **説明**: エンジンがオブジェクトをデータベースキャッシュに配置する際に空き空間を作成する必要が出た場合、データベースキャッシュからリリースするメモリの最小サイズ (バイト単位)。 このセレクタの目的はキャッシュからデータをリリースする時間を減らし、よりよりパフォーマンスを得ることにあります。キャッシュのサイズやデータベース中で処理されるデータのブロックサイズに応じてこの値を変更できます。 このセレクタが使用されないとデフォルトで、4Dは空間が必要になった時最低10%のキャッシュをアンロードします。
Cache writing mode	倍長整数	26	*** このセレクターは廃止されました ***
Character set	倍長整数	17	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Client Character set	倍長整数	24	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ17参照 **Description**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client HTTPS Port ID	倍長整数	40	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: 0 ～ 65535 **説明**: このセレクタは、クライアントマシンのWebサーバがSSLによるセキュアな接続（HTTPSプロトコル）で使用するTCPポート番号を、プログラムで変更するために使用できます。デフォルトの値は443（標準ポート番号）です。 このセレクタの動作はセレクタ39と同じですが、Web サーバとして使用されるすべてのクライアントマシンに適用されます。特定のクライアントマシンの設定だけを変更するのであれば、4Dリモートの環境設定ダイアログ画面を使用してください。
Client IP Address to listen	倍長整数	23	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ16参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Log Recording	倍長整数	45	**スコープ**: リモート4Dマシン **2セッション間で設定を保持**: No **とりうる値**: 0 または 1 から X (0 = 記録しない, 1 から X = ファイル名に付加するシーケンス値)。 **説明**: コマンドを実行した4Dクライアントマシンが実行した標準的なリクエスト（Webリクエストを除く）の記録を開始または停止します。デフォルト値は0（リクエストを記録しない）です。 クライアントマシンが実行したリクエストをログファイルに記録するよう、4Dに指示することができます。このメカニズムが有効になると、クライアントマシンのデータベースのローカルフォルダ内、Logsサブフォルダに2つのログファイルが作成されます。ファイルには"4DRequestsLog_X"と"4DRequestsLog_ProcessInfo_X"（Xはログのシーケンシャル番号）の名前が付けられます。4DRequestsLogファイルサイズが10MBに達するとそのファイルは閉じられ、インクリメントしたシーケンス番号を使用して新しいファイルが作成されます。もし同じ名前のファイルが存在する場合は置き換えられます。シーケンシャル番号の開始番号を、value引数を使用して指定できます。 このテキストファイルにはそれぞれのリクエストに関する様々な情報（時間、プロセス番号、ユーザ、リクエストサイズ、処理時間など）がシンプルな表形式のフォーマットで保存されます。この情報はアプリケーション開発フェーズや統計の目的で利用する場合に特に役立ちます。
Client Max Concurrent Web Proc	倍長整数	25	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ18参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Max Web requests size	倍長整数	21	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ27参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Maximum Web Process	倍長整数	20	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ7参照 **Description**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Minimum Web Process	倍長整数	19	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ6参照 **説明**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Port ID	倍長整数	22	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: セレクタ15参照 **Description**: このセレクタを使用して、Webサーバとして使用する4Dクライアントマシンの動作パラメタを指定できます。これらのセレクタを用いて指定された値は、Webサーバとして使用するすべての4D Clientマシンに対して適用されます。特定の4D Clientマシンに対してのみ値を指定したい場合には、4D Clientの環境設定ダイアログボックスを使用してください。
Client Server Port ID	倍長整数	35	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0～65535 **説明**: 4D Server が（4D Client に対して）データベースを公開するために使用されるTCPポート番号をプログラムで変更するために使用します。デフォルト値は19813 です。 この値を変更すれば、TCPプロトコルを使用して、複数の4D クライアント/サーバアプリケーションを同じマシンで同時に使用することができます。その場合、それぞれのアプリケーションごとに異なるポート番号を設定します。 公開ポート番号は、ストラクチャファイルに記録されています。ローカルモードの4Dで設定することもできますが、クライアント/サーバ環境でのみ考慮されます。 値を変更した場合、新しい設定が有効になるためには、アプリケーションを再起動する必要があります。
Client Web Log Recording	倍長整数	30	**スコープ**: すべての4Dリモートマシン **2セッション間で設定を保持**: Yes **とりうる値**: 0 = 記録しない (デフォルト), 1 = CLFフォーマットで記録, 2 = DLFフォーマットで記録, 3 = ELFフォーマットで記録, 4 = WLFフォーマットで記録 **説明**: すべてのクライアントマシンのWebサーバが受け取るWebリクエストの記録を開始または停止します。デフォルト値は0（リクエストを記録しない）です。 このセレクタの動作はセレクタ29と同じです。しかし対象はWebサーバとして使用されるすべての4Dクライアントマシンです。この場合、"logweb.txt"ファイルは4Dクライアントデータベースフォルダ (キャッシュフォルダ) のLogsサブフォルダに作成されます。特定のクライアントマシンにのみ設定を行いたい場合、4Dクライアントの環境設定を使用します。
Database Cache Size	倍長整数	9	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: - **説明**: カレントのデータベースメモリキャッシュサイズを取得するために使用します。戻り値はバイト単位です。 最大キャッシュサイズは環境設定の"データベース/データ管理"ページで設定できます。しかしデータベースキャッシュに実際に割り当てられる値は設定とシステムの現在のリソースに基づきます。このセレクタを使用すれば4Dによりデータベースに割り当てられた実際のメモリサイズを取得できます。 **警告**: ランゲージを使用してデータベースキャッシュメモリサイズを設定することはできません。言い換えればDatabase Cache SizeをSET DATABASE PARAMETERコマンドで使用することはできません。

Debug log recording	倍長整数	34	**Scope**: 4D application
			Kept between two sessions: No
			Description: Starts or stops the sequential recording of events occurring at the 4D programming level in the *4DDebugLog* file, which is automatically placed in the Logs subfolder of the database, next to the structure file. A new, more compact, tabbed text format is used in the event log file "4DDebugLog[_n].txt" starting with 4D v14 (where _n is the segment number of the file).
			Possible values: Longint containing a bit field: value = bit1(1)+bit2(2)+bit3(4)+bit4(8)+…).
			- Bit 1 (value 1) requests to enable the file (note that any other non-null value also enables it as well)
			- Bit 2 (value 2) requests call parameters to methods and commands.
			- Bit 3 (value 4) enables new tabbed format.
			- Bit 4 (value 8) disables immediate writing of each operation on disk (enabled by default). Immediate writing is slower but more effective, for example for investigating causes of a crash. If you disable this mode, the file contents are more compact and are generated more quickly.
			- Bit 5 (value 16) disables recording of plug-in calls (enabled by default).
			Execution times (in milliseconds) are always included in the file. The "< ms" value is displayed when an operation lasts less than one millisecond.
			Examples:
			SET DATABASE PARAMETER (34;1) // enables mode v13 file without parameters, with runtimes
			SET DATABASE PARAMETER (34;2) // enables mode v13 file with parameters and runtimes
			SET DATABASE PARAMETER (34;2+4) // enables file with v14 format, with parameters and runtimes
			SET DATABASE PARAMETER (34;0) // disables file
			To avoid having a file record too much information, you can restrict the 4D commands that are examined by using selector 80, Log Command list.
			This option can be enabled for any type of 4D application (4D all modes, 4D Server, 4D Volume Desktop), in interpreted or compiled mode.
			Note: This option is provided solely for the purpose of debugging and must not be put into production since it may lead to deterioration of the application performance and saturation of the hard disk. For more information about this format and on the use of the 4DDebugLog[_n].txt file, please contact the Technical Support of 4D Inc.

Diagnostic Log Recording	倍長整数	79	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **取りうる値**: 0 または 1 (0 = 記録しない, 1 = 記録する) **説明**: 4D診断ファイルの記録を開始または停止する。デフォルトの値は0で記録を行いません。 4Dは診断ファイルの中に内部的なアプリケーション処理に関連するイベントを継続的に記録することができます。このファイルに含まれる情報は4Dアプリケーション開発のために使用され、4D社の技術サポート担当により解析されます。このセレクターに1を渡すと、DatabaseName.txtという名称のファイルが自動でデータベースの**Logs**フォルダーに作成されるか、既に存在する場合は開かれます。このファイルのサイズが10MBに達するとそのファイルは閉じられ、DatabaseNameN.txtが生成されます (Nは連番)。 LOG EVENTコマンドを使用してカスタム情報をこのファイルに書き込むこともできます。
Direct2D Disabled	倍長整数	0	セレクター69 (Direct2D Status) 参照。

Direct2D Get active status	倍長整数	74	**注:** このセレクターはGet database parameterコマンドでのみ使用することができ、設定することはできません。
			説明: WindowsにおいてDirect2Dのアクティブな実装を返します。
			取りうる値: 0, 1, 2, 3, 4 または 5 (セレクター69の値参照)。返される値はDirect2Dが利用可能かどうか、およびハードウェア、OSによってサポートされるDirec2Dの品質に基づきます。
			例えば以下のコードを実行した場合、
			`SET DATABASE PARAMETER(Direct2D Status;Direct2D Hardware)`
			`$mode:=Get database parameter(Direct2D Get active status)`
			- Windows 7以降、システムがDirect2D互換のハードウェアを検知すると、$modeに1が設定されます。そうでなければ$modeは3に設定されます (ソフトウェアコンテキスト)。
			- Windows Vistaでは、システムがDirect2D互換のハードウェアを検知すると、$modeに1が設定されます。そうでなければ$modeは0に設定されます (Direct2D無効)。
			- Windows XPでは、$modeは常に0です (Direct2D非互換)。

Direct2D Hardware	倍長整数	1	セレクター69 (Direct2D Status) 参照。
Direct2D Hardware SVG and Editors	倍長整数	2	セレクター69 (Direct2D Status) 参照。
Direct2D Mixed	倍長整数	5	セレクター69 (Direct2D Status) 参照。
Direct2D Software	倍長整数	3	セレクター69 (Direct2D Status) 参照。
Direct2D Software SVG and Editors	倍長整数	4	セレクター69 (Direct2D Status) 参照。
Direct2D Status	倍長整数	69	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **説明**: WindowsにおけるDirect2D実装のアクティベーションモード **取りうる値**: 以下の定数のいずれか (デフォルトでモード5): Direct2D Disabled (0): Direct2Dモードは無効であり、データベースは過去のモード (GDI/GDIPlus) で動作する。 Direct2D Hardware (1): 4Dアプリケーション全体でDirect2Dグラフィックハードウェアコンテキストを使用する。このコンテキストが利用できない場合、Direct2Dグラフィックソフトウェアコンテキストを使用 (Vistaを除く。Vistaではパフォーマンスのために GDI/GDIPlusモードが使用されます)。 Direct2D Hardware SVG and Editors (2): SVG、コード、およびストラクチャーエディターでDirect2Dグラフィックハードウェアコンテキストを使用する。このコンテキストが利用できない場合、Direct2Dグラフィックソフトウェアコンテキストを使用 (Vistaを除く。VistaではパフォーマンスのためにGDI/GDIPlusモードが使用されます)。 Direct2D Software (3): Windows 7以降、4Dアプリケーション全体でDirect2Dグラフィックソフトウェアコンテキストを利用。VistaではパフォーマンスのためにGDI/GDIPlusモードが使用されます。 Direct2D Software SVG and Editors (4): Windows 7以降、SVG、コード、およびストラクチャーエディターのDirect2Dグラフィックソフトウェアコンテキストを使用する。VistaではパフォーマンスのためにGDI/GDI/GDIPlusモードが使用されます。 Direct2D Mixed (5) (デフォルトモード): Windows 7以降、SVG、コード、およびストラクチャーエディターでDirect2Dグラフィックハードウェアコンテキストを使用する。4Dアプリケーションの他の部分ではDirect2Dグラフィックソフトウェアコンテキストを使用する。VistaではパフォーマンスのためGDI/GDIPlusモードを使用します。
HTTP Compression Level	倍長整数	50	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
HTTP Compression Threshold	倍長整数	51	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
HTTPS Port ID	倍長整数	39	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Idle Connections Timeout	倍長整数	54	**スコープ**: 値が負数なら4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**: 持続時間を秒で表す値。値は正数 (新規接続) または負数 (既存の接続)をとることができます。デフォルト値は4D v11 SQLの場合0 (タイムアウトなし) で、4D v12の場合20です。 **説明**: この引数を使用して、4DデータベースエンジンとSQLエンジン両方への動きのない接続の最大時間 (タイムアウト) を設定できます。この設定は、コマンドが実行されたマシンにより開かれたすべての接続に適用されます。 動作していない接続がこの制限時間に達すると、接続は自動でスタンバイ状態に置かれます。つまりクライアント/サーバセッションがフリーズされ、ネットワークソケットが閉じられます。この動作はユーザに対し完全に透過的です。スタンバイ状態の接続でリクエストが開始されると、ソケットが自動で再び開かれ、クライアント/サーバセッションが再び有効になります。 この設定によりサーバのリソースを節約できます。スタンバイ状態の接続はソケットを閉じ、サーバ上のプロセスを解放します。他方これにより、ファイアウォールがアイドルなソケットを閉じてしまうことに伴い接続が失われることを避けることができます。このためには、アイドル接続のタイムアウト値はファイアウォールのタイムアウト値よりも小さくなくてはなりません。 valueに正数を渡すと、設定はすべてのプロセスのすべての新規接続に適用されます。負数を渡すと、設定はカレントプロセスの開かれた接続に適用されます。0を渡すと、アイドル接続のタイムアウトは行われません。 4D v11 SQLの場合、このパラメターはサーバ上でのみ考慮されます。 4D v12の場合、このパラメターはサーバーおよびクライアント両側で設定できます。2つの異なる間隔を設定すると、短いほうが使用されます。通常この値を変更する必要はありません。
Index Compacting	倍長整数	4	*** このセレクターは廃止されました ***
Invert Objects	倍長整数	37	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0, 1 または 2 (0 = モードは無効, 1 = 自動モード, 2 = モードが有効) **説明**: Right-to-left 言語のWindowsでデータベースが表示されるとき、アプリケーションモードでフォームやオブジェクト、メニューバーなどを反転させるために使用される、"オブジェクト反転"モードを設定します。このモードはアプリケーション環境設定のデータベース/インターナショナルページで変更できます。 • 0 に設定した場合、システム設定に関係なく、モードは無効です（環境設定でいいえにするのと同じ）。 • 1 に設定した場合、システム設定に応じ、モードが有効または無効になります（環境設定を自動にするのと同じ）。 • 2 に設定した場合、システム設定に関係なく、モードは有効です（環境設定をはいにするのと同じ）。 詳細は4DのDesign Referenceマニュアルを参照してください。
IP Address to listen	倍長整数	16	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
JSON use local time	倍長整数	85	**Scope**: 4D application **Kept between two sessions**: No **Possible values**: 0 = ignore local time zone, 1 (default) = take time zone into account. **Description**: By default, 4D dates converted to JSON format take the local time zone into account. For example, converting the date !23/08/2013! gives you "2013-08-22T22:00:00Z" in JSON format when the operation is performed in France during Daylight Savings Time (GMT+2). This principle conforms to the standard operation of JavaScript. This can be a source of errors when you want to send JSON date values to someone in a different time zone. This is the case, for example, when you export a table using Selection to JSON in France that is meant to be reimported in the US using JSON TO SELECTION. By default, since dates are re-interpreted in each time zone, the values stored in the database will be different. In this case, you can modify the conversion mode for dates so that they do not take the time zone into account by passing 0 in this selector. Converting the date !23/08/2013! will then give you "2013-08-23T00:00:00Z" in all cases.
Log Command list	文字列	80	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **取りうる値**: 記録する4Dコマンドの番号リスト。型は文字列で各コマンド番号をセミコロンで区切ります。"all"を渡すとすべてのコマンドが記録され、"" (空の文字列) を渡すとなにも記録されません。 **説明**: デバッグファイルに記録する4Dコマンドのリスト (セレクター 34, Debug Log Recording参照)。デフォルトではすべての4Dコマンドが記録されます。 このセレクターを使用すれば、記録に残したい4Dコマンドを指定することで、デバッグファイルに保存される情報の量を制限することができます。例えば以下のようにコードを記述できます: `SET DATABASE PARAMETER(Log Command list;"277;341")`

```
// QUERY および QUERY SELECTION コマンドのみを記録する
```

Max Concurrent Web Processes	倍長整数	18	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Maximum Temporary Memory Size	倍長整数	61	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**:正の倍長整数値 **説明**: 4D がそれぞれのプロセスに割り当てることのできる一時的なメモリの最大サイズ (MB)。デフォルトで値は 0 (最大サイズの設定なし) です。 4D はインデックスやソート処理のために特別な一時的メモリを使用します。このメモリは大量の処理を行う間、" 標準" キャッシュメモリの保護を意図したものです。これは必要な時にのみ有効になります。デフォルトで、一時的なメモリのサイズは、(システムメモリ設定に基づく) 利用可能なリソースによってのみ制限されます。 このメカニズムはほとんどのアプリケーションで適しています。しかし特定の特別なコンテキスト、特に同時に多数のシーケンシャルソートを行うようなサーバ/ クライアントアプリケーションでは、一時的なメモリのサイズが、システムが不安定になるほどに致命的に増加するかもしれません。このような場合は、一時的メモリの最大サイズを設定することで、アプリケーションが正しく動作するようにできます。その代わり、実行速度に影響が出ます。プロセスに対する最大サイズに達すると、4D はディスクファイルを使用し、そのために処理が遅くなります。 先のようなケースの場合、だいたい50 MB が一般的なサイズとしてよいと思われます。しかし適切な値はアプリケーションの特性、そして実際の環境でのテスト結果に基づき決定されるべきです。
Maximum Web Process	倍長整数	7	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **とりうる値**: 0 -> 32 767 **説明**: ローカルモードの4Dならびに4D Serverを使用した場合に、非コンテキストモードで保持するWebプロセスの最大数。デフォルト値は10。 非コンテキストモードでのWebサーバの反応を良くするため、4Dは5秒間Webプロセスを遅延させ、将来やってくるかもしれないHTTPクエリを処理する際、それを再利用します。パフォーマンスの観点では、クエリごとに新しいプロセスを作成するよりも、再利用したほうが実際に有利です。Webプロセスが再利用されると、さらに5秒間遅延されます。Webプロセスが最大数に達するとプロセスがアボートされます。5秒の遅延以内にWebプロセスがクエリを受け取らない場合、Webプロセスの最小数を下回らなければ、プロセスはアボートされます。最小数未満になる場合、プロセスは再度遅延されます。 これらの引数は、リクエスト数や利用可能なメモリ、その他のパラメタに応じて、Webサーバの動作を調整できるようにするものです。
Maximum Web requests size	倍長整数	27	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。
Minimum Web Process	倍長整数	6	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **とりうる値**: 0 -> 32 767 **説明**: ローカルモードの4Dならびに4D Serverを使用した場合に、非コンテキストモードで保持するWebプロセスの最小数。デフォルト値は0（下記参照）。
Order By Formula On Server	倍長整数	47	**スコープ**: カレントテーブルおよびプロセス **2セッション間で設定を保持**: No **とりうる値**: 0 (データベース設定を使用), 1 (クライアント上で実行) または 2 (サーバ上で実行) **説明**: 引数に渡されたtableに対して実行されるORDER BY FORMULAコマンドの実行場所。 クライアント/サーバモードでデータベースを使用するとき、ORDER BY FORMULAコマンドをサーバ上またはクライアント上で実行させることができます。このセレクタを使用して、このコマンドの実行場所 (サーバまたはクライアント) を指定できます。このモードはデータベース環境設定でも設定できます。詳細はセレクタ46、Query By Formula On Serverの説明を参照してください。 **Note:** "SQLタイプ"のJOINを有効にしたい場合 (QUERY BY FORMULA Joins (49) セレクタ参照) は、フォーミュラがレコードにアクセスすることができるよう、常にフォーミュラをサーバ上で実行しなければなりません。このコンテキストでは、フォーミュ ラはいかなるメソッド呼び出しも含んではいけないことに注意してください。そうでない場合、自動でリモートマシンでの実行に切り替わります。
PHP Interpreter IP address	倍長整数	55	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: "nnn.nnn.nnn.nnn" (例 "127.0.0.1") のようなフォーマット文字列 **説明**: FastCGI を経由してPHPインタプリタと通信を行うために、4D がローカルで使用するIP アドレス。デフォルトで値は"127.0.0.1" です。このアドレスは4D が配置されているマシンに対応しなければなりません。このパラメタはデータベース設定を使用してすべてのマシン用にグローバルに設定できます PHPインタプリタに関する詳細は*Design Reference*マニュアルを参照してください。
PHP Interpreter port	倍長整数	56	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: 正の倍長整数値。デフォルト値は8002。 **説明**: 4DのPHPインタプリタが使用するTCPポート番号。データベース設定でこの値をすべてのマシン用にグローバルに変更できます。PHPインタープリタに関する詳細は*Design Reference* マニュアルを参照してください。
PHP Max requests	倍長整数	58	**スコープ**: 4D application **2セッション間で設定を保持**: No **値**: 正の倍長整数地。デフォルト値は500。 **説明**: PHP インタプリタが受け入れるリクエストの最大数。この最大値に達すると、インタプリタは"server busy"タイプのエラーを返します。セキュリティおよびパフォーマンスのため、この値を変更できます。データベース設定を使用してすべてのマシン用にグローバルに設定を変更できます。このパラメタに関する詳細はFastCGIPHPのドキュメントを参照してください。 **Note:** 4D側では、これらの引数は動的に適用されます。設定を有効にするために4Dを終了する必要はありません。他方、PHPインタプリタが既に起動されている場合、これらの設定を有効にするためにはインタプリタを再起動しなければなりません。
PHP Number of children	倍長整数	57	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: 正の倍長整数値。デフォルト値は5。 **説明**: 4DのPHPインタプリタがローカルで作成し、管理する子プロセスの数。最適化の目的で、スクリプト実行リクエストを処理するために、PHPインタプリタは" 子プロセス" と呼ばれるシステムプロセスのセット (プール) を作成、使用します。アプリケーションのニーズに基づき、子プロセス数の数を変更できます。データベース設定を使用してこの値をすべてのマシン用にグローバルに設定できます。PHPインタプリタに関する詳細は*Design Reference* マニュアルを参照してください。 **Note:** Mac OS では、すべての子プロセスは同じポートを共有します。Windows では、それぞれの子プロセスが個別のポート番号を使用します。最初の番号はPHP インタプリタ用に設定された番号で、他の子プロセスは最初の番号をインクリメントします。例えばデフォルトポート番号が8002で、5個の子プロセスを起動すると、ポート8002から8006が使用されます。
PHP Use external interpreter	倍長整数	60	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: No **値**: 0 = 内部インタプリタを使用, 1 = 外部インタプリタを使用 説明: 4D のPHPリクエストを4Dの内部インタプリタに送信するか、外部インタプリタに送信するかを指定する値。デフォルトで値は0 (4Dのインタプリタを使用) です。独自にインストールしたPHPインタプリタを使用したい場合、例えば追加のモジュールを使用したり特定の設定を使用したい場合は、valueに1を渡します。この場合、4DはPHPリクエストの場合でも内部インタプリタを起動しません。 カスタムPHPインタプリタはFastCGIでコンパイルされ、4Dエンジンと同じマシン上に配置されていなければなりません。この場合、開発者がインタプリタを完全に管理しなければならない点に留意してください。4Dはインタプリタを開始したり停止したりしません。 データベース設定を使用してこの値をすべてのマシン用にグローバルに設定できます。
Port ID	倍長整数	15	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: No **説明**: この引数を使用して、ローカルモードの4Dおよび4D Serverによる4D Webサーバが使用するTCPポートをオンザフライで変更することができます。デフォルト値は80で、この値は環境設定ダイアログボックスの"Web/設定"ページで設定できます。TCP Port Numbersテーマの定数をvalue引数に使用できます。 Port IDセレクタは、コンパイルして4D Desktopを組み込んだ4D Webサーバで役立ちます (この場合、デザインモードへのアクセス手段がありません)。TCPポートIDに関する詳細はWebサーバー設定を参照してください。
QUERY BY FORMULA Joins	倍長整数	49	**スコープ**: カレントプロセス **2セッション間で設定を保持**: No **とりうる値**: 0 (データベース設定を使用), 1 (常に自動リレーションを使用) または 2 (可能ならSQL JOINを使用) **説明**: "SQL JOIN"の利用に関連する、QUERY BY FORMULAとQUERY SELECTION BY FORMULAコマンドの動作モード。 4D v11 SQLのバージョン11.2以降で作成されたデータベースでは、これらのコマンドはSQL JOINモデルに基づくJOINを実行します。このメカニズムを使用して、(以前のバージョンでは必要な条件だった) 自動リレーションで接続されていない他のテーブルに対して実行されたクエリに基づき、テーブルのセレクションを変更できます。 QUERY BY FORMULA Joinsセレクタで、カレントプロセスの、フォーミュラによるクエリの動作モードを指定できます: • 0: データベースの現在の設定を使用する (デフォルト値)。4D v11 SQLのバージョン11.2以降で作成されたデータベースでは、"SQL JOIN"はフォーミュラによるクエリに対し常に有効です。変換されたデータベースでは、互換性のためデフォルトでこのメカニズムは有効になっていません。しかし環境設定で実装できます。 • 1: 常に自動リレーションを使用 (= 以前のバージョンの4Dの動作)。このモードでは、他のテーブルに対して行われたクエリに基づきテーブルのセレクションを作成するためには、リレーションが必要です。4Dは"SQL JOIN"を行いません。 • 2: 可能であればSQL JOINを使用 (= 4D v11 SQLのバージョン11.2以降で作成されたデータベースのデフォルト動作)。このモードでは、フォーミュラが適合する限り、4Dはフォーミュラによるクエリに"SQL JOIN"を使用します (2つの例外があります。QUERY BY FORMULAやQUERY SELECTION BY FORMULA コマンドの説明を参照)。 **Note:** 4Dのリモートモードにおいて、"SQL JOIN"はフォーミュラがサーバ上で実行されているときにのみ使用できます (レコードへのアクセスが必要なため)。フォーミュラの実行場所を設定するには、セレクタ46と47を参照してください。
Query By Formula On Server	倍長整数	46	**スコープ**: カレントテーブルおよびプロセス **2セッション間で設定を保持**: No **とりうる値**: 0 (データベース設定を使用), 1 (クライアント上で実行) または 2 (サーバ上で実行) **説明**: 引数に渡されたtableに対して実行されるQUERY BY FORMULAやQUERY SELECTION BY FORMULAコマンドの実行場所。 クライアント/サーバモードでデータベースを使用するとき、フォーミュラを使用したクエリをサーバ上またはクライアント上で実行させることができます: • 4D v11 SQLで作成されたデータベースでは、これらのコマンドはサーバ上で実行されます。 • 変換されたデータベースでは、これらのコマンドは、以前のバージョンの4Dと同様、クライアントマシン上で実行されます。 • 変換されたデータベースでは、環境設定のアプリケーション/互換性ページで、これらのコマンドの実行場所をグローバルに変更できます。 この実行場所の違いは、アプリケーションのパフォーマンス (通常サーバ上で実行したほうが早い) だけでなく、プログラミングにも影響します。実際フォーミュラの部品の値 (特にメソッドから呼ばれる変数) は、実行コンテキストにより異なります。このセレクタを使用して開発者は、アプリケーションの動作を適応させられます。 value 引数に0を渡すと、フォーミュラを使用するクエリの実行場所は、データベースの設定に基づきます: 4D v11 SQLで作成されたデータベースでは、これらのコマンドはサーバ上で実行されます。変換されたデータベースでは、データベース環境設定に基づき、クライアントマシンまたはサーバマシンで実行されます。valueに1または2を渡すと、これらのコマンドの実行場所をクライアントマシンまたはサーバマシンに強制できます。 例題4を参照してください。 **Note:** "SQLタイプ"のJOINを有効にしたい場合 (QUERY BY FORMULA Joins (49) セレクタ参照) は、フォーミュラがレコードにアクセスすることができるよう、常にフォーミュラをサーバ上で実行しなければなりません。このコンテキストでは、フォーミュラはいかなるメソッド呼び出しも含んではいけないことに注意してください。そうでない場合、自動でリモートマシンでの実行に切り替わります。
QuickTime support	倍長整数	82	**Scope**: 4D application **Kept between two sessions**: No **Possible values**: 0 (default) = QuickTime disabled, 1 = QuickTime activated. **Description**: In 4D starting with v14, by default QuickTime codecs are no longer supported. For compatibility, you can use this selector to re-enable them in your application. However, you should note that in future versions of 4D, QuickTime support is permanently removed.
Seq Access Optimization	倍長整数	2	*** このセレクターは廃止されました ***
Seq Distinct Values Ratio	倍長整数	3	*** このセレクターは廃止されました ***
Seq Order Ratio	倍長整数	1	*** このセレクターは廃止されました ***
Seq Query Select Ratio	倍長整数	5	*** このセレクターは廃止されました ***
Server Base Process Stack Size	倍長整数	53	**スコープ**: 4D Server **2セッション間で設定を保持**: No **とりうる値**:正の倍長整数 **説明**: サーバ上のプリエンプティブシステムプロセス毎に割り当てるスタックのサイズ (バイト単位) です。デフォルトでこの値は1,000,000 (1 MB) です。 プリエンプティブシステムプロセスはメインの4D クライアントプロセスを制御するためにロードされます。デフォルトでそれぞれのプリエンプティブプロセスに割り当てられるサイズはおよその場合最適なサイズですが、何百ものプロセスが作成されるようなケースではこのサイズが適切かどうか検討する必要が出てくるかもしれません。 データベースが実行する処理がそれを許す限り、最適化の目的でこのサイズを大幅に減らすことができます (例えばデータベースで大量のレコードの並び替えなどを行わない場合)。512 や256 KB でさえも設定可能です。スタックサイズを小さくしすぎることは致命的であり、4D Server の動作に害を及ぼすことになるので注意してください。このパラメタの設定は注意を持って行い、データベースの利用状況 (レコード数や行う処理など) を考慮しながら行わなければなりません。 このパラメタの設定を行うには、On Server Startup データベースメソッドなどを使用してサーバ上でコマンドが実行されなければなりません。
Spellchecker	倍長整数	81	**Scope**: 4D application **Kept between two sessions**: No **Possible values**: 0 (default) = native OS X spellchecker (Hunspell disabled), 1 = Hunspell spellcheck enabled. **Description**: Enables the Hunspell spellcheck under OS X. By default, the native spellchecker is enabled on this platform. You may prefer to use the Hunspell spellcheck, for example, in order to unify the interface for your cross-platform applications (under Windows, only the Hunspell spellcheck is available). For more information, refer to Support of Hunspell dictionaries.
SQL Autocommit	倍長整数	43	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0 (無効) または 1 (有効) **説明**: SQLの自動コミットモードを有効または無効にするために使用します。デフォルトは 0 (無効モード) です。 自動コミットモードは、データベースの参照整合性を強化するために使用されます。このモードが有効の時、すべてのSELECT, INSERT, UPDATE そして DELETE (SIUD) クエリは、これらがトランザクション内で実行されていない場合、自動でアドホックなトランザクションに含められます。このモードはデータベースの環境設定でも設定できます。
SQL Engine Case Sensitivity	倍長整数	44	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0 (大文字小文字を考慮しない) または 1 (考慮する) **説明**: SQLエンジンが文字列比較を行う際に、大文字と小文字の違いを考慮させるかどうかを設定します。 デフォルトで値は1 (大文字小文字を考慮する) です。SQLエンジンは文字列比較 (並び替えやクエリ) の際に大文字と小文字を異なる文字として扱います。例えば"ABC"= "ABC"ですが"ABC" # "Abc"です。SQLエンジンと4Dエンジンの動作をそろえたいなど特定の場合では、大文字と小文字を区別しない文字列比較 ("ABC"="Abc") を使用できます。 このオプションはアプリケーション環境設定の SQL/設定ページで設定できます。
SSL Cipher List	文字列	64	**スコープ**: 4Dアプリケーション **2セッション間で設定を保持**: No **とりうる値**: コロンで区切られた一連の文字列 (例 "RC4-MD5:RC4-64-MD5:....") **説明**: 暗号リストはSSLプロトコルのために4Dが使用します。例えばこのセレクターを使用してSSL 3.0暗号化アルゴリズムを実装でき、そしてSSL 2.0による接続を拒否できます。 この設定はアプリケーション全体に適用されます (HTTP サーバーやSQLサーバー、およびSSL プロトコルを使用するすべての4D の機能に関連) が、その効果は一時的であり、セッションをまたいでは保持されません。 暗号リストが変更されると、新しい設定を有効にするために、関連するサーバーを再起動しなければなりません。 暗号リストを (SLIファイルに恒久的に格納された) デフォルト値に再設定するには、value引数に空の文字列 ("") を渡してSET DATABASE PARAMETERコマンドを呼び出します。 デフォルトで、4DはRC4暗号化アルゴリズムを使用します。(より最新の) AESアルゴリズムを使用するには、value引数に以下の文字列を渡します: "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH" **Note**: Get database parameterコマンドで暗号リストはオプションのstringValue引数に返され、戻り値は常に0となります。
Table Sequence Number	倍長整数	31	**スコープ**: 4D アプリケーション **2セッション間で設定を保持**: Yes **とりうる値**: 任意の倍長整数値 **説明**: このセレクタは、引数に渡したテーブルのレコードの、カレントのユニーク番号を取得あるいは設定するために使用します。"カレントの数値"とは"最後に使用された数値"を意味します。SET DATABASE PARAMETERコマンドを使用してこの値を変更すると、渡された値+1の番号を使用して次のレコードが作成されます。この新しい番号は、Sequence numberコマンドによって返される、さらにはストラクチャエディタやSQLで自動インクリメントが設定されたフィールドに返される番号です。 デフォルトで、この固有の番号は4Dが設定し、レコードの作成順に対応します。詳細はSequence numberコマンドのドキュメントを参照してください。
TCP_NODELAY	倍長整数	33	*** このセレクターは廃止されました ***
Temporary memory size	倍長整数	42	*** このセレクターは廃止されました ***
Unicode mode	倍長整数	41	**スコープ**: データベース **2セッション間で設定を保持**: Yes **とりうる値**: 0 (互換モード) または 1 (Unicodeモード) **説明**: カレントデータベースの文字セットに関連する動作モード。4DはUnicode文字セットをサポートしますが、(Mac ASCII文字セットに基づく)"互換"モードで動作させることができます。デフォルトで、変換されたデータベースは互換モード (0) で、バージョン11以降で作成されたデータベースはUnicodeモードで実行されます。実行モードは環境設定のオプションでコントロールでき、またこのセレクタを使用して読みだしたり、(テスト目的で) 変更したりできます。このオプションを変更した場合、それを有効にするにはデータベースを再起動しなければなりません。コンポーネント内部ではこの値を変更できないことに留意してください。読み出しのみが可能です。
Web Conversion Mode	倍長整数	8	**スコープ**: カレントプロセス **2セッション間で設定を保持**: No **とりうる値**: 0, 1, 2 または 3 • (デフォルトモード) ブラウザが対応する場合、HTML 4.0フォーマットに変換。そうでなければHTML 3.2フォーマットと配列を使用。 • 1 = 6.0.x変換モード • 2 = 6.5変換モード • 3 = HTML 4.0フォーマット + CSS-P (バージョン6.5.2より) に変換。 **説明**: ローカルモードの4Dならびに4D Serverで使用するWeb用の4Dフォーム変換モード。デフォルトで、4D WebサーバはCSS1を使用し、4Dで表示される4Dフォームと同様のHTMLページを生成します。この機能では、バージョン6.7より前の4Dで作成されたデータベースに関しては、フォームが正しく変換されない可能性があります。このため、フォーム変換モードを設定する必要があるかもしれません。 このモードは、SET DATABASE PARAMETERが呼び出されたプロセス(Webコンテキスト)に対してのみ設定されます。このコマンドは、メソッド内で呼び出してデータベースのすべてのフォームを確実に統一したり、あるいは特定のフォームを表示する前にのみ呼び出すこともできます。このコマンドは、コンテストモード、またはWebプロセス以外の場所から呼び出すと、何も行いません。
Web Log Recording	倍長整数	29	**説明**: 廃止 (互換性のために保持)。HTTPサーバー設定を変更するためにはWEB SET OPTIONとWEB GET OPTIONコマンドを使用してください。

Date Display Formats

定数	型	値	コメント
Blank if null date	倍長整数	100	0の代わりに""
Date RFC 1123	倍長整数	10	Fri, 10 Sep 2010 13:07:20 GMT
Internal date abbreviated	倍長整数	6	Dec 29, 2006
Internal date long	倍長整数	5	December 29, 2006
Internal date short	倍長整数	7	2006/12/29
Internal date short special	倍長整数	4	06/12/29 (しかし 1986/12/29 または 2096/12/29)
ISO Date	倍長整数	8	2006-12-29T00:00:00
ISO Date GMT	倍長整数	9	2010-09-13T16:11:53Z
System date abbreviated	倍長整数	2	Sun, Dec 29, 2006
System date long	倍長整数	3	Sunday, December 29, 2006
System date short	倍長整数	1	12/29/2006

Days and Months

定数	型	値	コメント
April	倍長整数	4	
August	倍長整数	8	
December	倍長整数	12	
February	倍長整数	2	
Friday	倍長整数	6	
January	倍長整数	1	
July	倍長整数	7	
June	倍長整数	6	
March	倍長整数	3	
May	倍長整数	5	
Monday	倍長整数	2	
November	倍長整数	11	
October	倍長整数	10	
Saturday	倍長整数	7	
September	倍長整数	9	
Sunday	倍長整数	1	
Thursday	倍長整数	5	
Tuesday	倍長整数	3	
Wednesday	倍長整数	4	

Design Object Access

定数	型	値	コメント
Attribute Executed on server	倍長整数	8	"サーバー上で実行"オプションに対応
Attribute folder name	倍長整数	1024	メソッドのためのフォルダ名(「フォルダ」属性)。この定数を渡した場合、フォルダ名を attribValue に渡す必要があります。 • この名前が有効なフォルダに対応する場合、メソッドはその親フォルダに置かれます。 • フォルダが存在しない場合、コマンドは親フォルダの階層は何も変更しません。 • 空の文字列を渡した場合、メソッドはルート階層に置かれます。
Attribute Invisible	倍長整数	1	"隠す"オプションに対応
Attribute Published SOAP	倍長整数	3	"Webサービスとして提供"オプションに対応
Attribute Published SQL	倍長整数	7	"SQL利用可"オプションに対応
Attribute Published Web	倍長整数	2	"4DタグおよびURL (4DACTION...) で利用可"オプションに対応
Attribute Published WSDL	倍長整数	4	"WSDLで公開する"オプションに対応
Attribute Shared	倍長整数	5	"コンポーネントとホストデータベースで共有する"オプションに対応
On object locked abort	倍長整数	0	オブジェクトのロードが中断された (デフォルト動作)
On object locked confirm	倍長整数	2	再試行するか中断するか選択できるようにするためダイアログを表示します。リモートモードではこのオプションはサポートされません (ロードは中断されます)。
On object locked retry	倍長整数	1	オブジェクトがリリースされるまでオブジェクトのロードを試行します。
Path All objects	倍長整数	31	データベースのすべてのメソッドのパス
Path Database method	倍長整数	2	指定したデータベースメソッド名。以下のメソッドのリスト: *[databaseMethod]/onStartup* *[databaseMethod]/onExit* *[databaseMethod]/onDrop* *[databaseMethod]/onBackupStartup* *[databaseMethod]/onBackupShutdown* *[databaseMethod]/onWebConnection* *[databaseMethod]/onWebAuthentication* *[databaseMethod]/onWebSessionSuspend* *[databaseMethod]/onServerStartup* *[databaseMethod]/onServerShutdown* *[databaseMethod]/onServerOpenConnexion* *[databaseMethod]/onServerCloseConnection* *[databaseMethod]/onSystemEvent* *[databaseMethod]/onSqlAuthentication*
Path Project form	倍長整数	4	プロジェクトフォームメソッドとすべてのフォームオブジェクトメソッドのパス。例: *[projectForm]/myForm/{formMethod}* *[projectForm]/myForm/button1* *[projectForm]/myForm/my%2list* *[projectForm]/myForm/button1*
Path Project method	倍長整数	1	メソッド名。 例: *MyProjectMethod*
Path Table form	倍長整数	16	テーブルフォームメソッドとすべてのフォームオブジェクトメソッド。例: *[tableForm]/table_1/Form1/{formMethod}* *[tableForm]/table_1/Form1/button1* *[tableForm]/table_1/Form1/my%2list* *[tableForm]/table_2/Form1/my%2list*
Path Trigger	倍長整数	8	データベーストリガーのパス。例: *[trigger]/table_1* *[trigger]/table_2*

Dictionaries

Compatibility note: これらの定数は"コーディアル"辞書の数字と対応していますが、この辞書は4D v14以降サポートされなくなりました。これらの定数は互換性のために残されています(内部では同等のハンスペル辞書が使用されます)。

定数	型	値	コメント
English Dictionary	倍長整数	69632	
French Dictionary	倍長整数	262144	
German Dictionary	倍長整数	131584	
Norwegian Dictionary	倍長整数	589824	
Spanish Dictionary	倍長整数	196608	

Digest Type

定数	型	値	コメント
4D digest	倍長整数	2	4D内部のアルゴリズムを使用
MD5 Digest	倍長整数	0	MD5アルゴリズムを使用
SHA1 Digest	倍長整数	1	SHA-1アルゴリズムを使用

Euro Currencies

定数	型	値	コメント
Austrian Schilling	文字列	ATS	
Belgian Franc	文字列	BEF	
Deutschemark	文字列	DEM	
Euro	文字列	EUR	
Finnish Markka	文字列	FIM	
French Franc	文字列	FRF	
Greek Drachma	文字列	GRD	
Irish Pound	文字列	IEP	
Italian Lira	文字列	ITL	
Luxembourg Franc	文字列	LUF	
Netherlands Guilder	文字列	NLG	
Portuguese Escudo	文字列	PTE	
Spanish Peseta	文字列	ESP	

Events (Modifiers)

定数	型	値	コメント
Activate window bit	倍長整数	0	
Activate window mask	倍長整数	1	
Caps Lock key bit	倍長整数	10	
Caps Lock key mask	倍長整数	1024	
Command key bit	倍長整数	8	
Command key mask	倍長整数	256	Windows = Ctrlキー、Mac OS = Commandキー
Control key bit	倍長整数	12	
Control key mask	倍長整数	4096	Mac OSのみ
Mouse button bit	倍長整数	7	
Mouse button mask	倍長整数	128	
Option key bit	倍長整数	11	
Option key mask	倍長整数	2048	Windows = Altキー、Mac OS = Optionキー
Right control key bit	倍長整数	15	
Right control key mask	倍長整数	32768	
Right option key bit	倍長整数	14	
Right option key mask	倍長整数	16384	
Right shift key bit	倍長整数	13	
Right shift key mask	倍長整数	8192	
Shift key bit	倍長整数	9	
Shift key mask	倍長整数	512	WindowsおよびMac OS

Events (What)

定数	型	値	コメント
Activate event	倍長整数	8	
Auto key event	倍長整数	5	
Disk event	倍長整数	7	
Key down event	倍長整数	3	
Key up event	倍長整数	4	
Mouse down event	倍長整数	1	
Mouse up event	倍長整数	2	
Null event	倍長整数	0	
Operating system event	倍長整数	15	
Update event	倍長整数	6	

Expressions

定数	型	値	コメント
MAXINT	倍長整数	32767	
MAXLONG	倍長整数	2147483647	
MAXTEXTLENBEFOREV11	倍長整数	32000	

Field and Variable Types

定数	型	値	コメント
Array 2D	倍長整数	13	
Boolean array	倍長整数	22	
Date array	倍長整数	17	
Integer array	倍長整数	15	
Is Alpha Field	倍長整数	0	
Is BLOB	倍長整数	30	
Is Boolean	倍長整数	6	
Is Date	倍長整数	4	
Is Float	倍長整数	35	
Is Integer	倍長整数	8	
Is Integer 64 bits	倍長整数	25	
Is JSON null	倍長整数	255	
Is LongInt	倍長整数	9	
Is object	倍長整数	38	
Is Picture	倍長整数	3	
Is Pointer	倍長整数	23	
Is Real	倍長整数	1	
Is String Var	倍長整数	24	
Is Subtable	倍長整数	7	
Is Text	倍長整数	2	
Is Time	倍長整数	11	
Is Undefined	倍長整数	5	
LongInt array	倍長整数	16	
Object array	倍長整数	39	
Picture array	倍長整数	19	
Pointer array	倍長整数	20	
Real array	倍長整数	14	
String array	倍長整数	21	
Text array	倍長整数	18	

Find Window

定数	型	値	コメント
In contents	倍長整数	3	Platform: Mac OS and Windows
In drag	倍長整数	4	Platform: Mac OS
In go away	倍長整数	6	Platform: Mac OS
In grow	倍長整数	5	Platform: Mac OS
In menu bar	倍長整数	1	Platform: Mac OS
In system window	倍長整数	2	Platform: Mac OS
In zoom box	倍長整数	8	Platform: Mac OS

Font Styles

Constants prefixed with _O_ are obsolete and must no longer be used.

定数	型	値	コメント
_O_Condensed	倍長整数	32	
_O_Extended	倍長整数	64	
_O_Outline	倍長整数	8	
_O_Shadow	倍長整数	16	
Automatic style sheet	文字列	__automatic__	
Bold	倍長整数	1	
Bold and Italic	倍長整数	3	
Bold and Underline	倍長整数	5	
Italic	倍長整数	2	
Italic and Underline	倍長整数	6	
Plain	倍長整数	0	
Underline	倍長整数	4	

Font Type List

定数	型	値	コメント
Favorite fonts	倍長整数	1	fonts にはお気に入りのフォント(マシン上で最もよく使われているフォント)のリストが返されます。 - Windows 環境下では、Windowsコントロールパネル内のアクティブなフォントファミリー名のリストが表示されます。 - OS X 環境下では、コントロールパネル内の "com.apple.Favorites" コレクションのフォントファミリー名のリストが表示されます。英語では"Favorites" 、フランス語では "Favoris" 、ドイツ語では"Favoriten" というように名前がついています。このコレクションは、ユーザーがお気に入りのフォントを何も追加していない場合には空であることがあります。
Recent fonts	倍長整数	2	fonts contains the list of recent fonts (the ones used during the 4D session). This list is used in particular by the .
System fonts	倍長整数	0	fonts contains the list of all the system fonts. Default option when listType is omitted.

Form Area

定数	型	値	コメント
Form Break0	倍長整数	300	
Form Break1	倍長整数	301	
Form Break2	倍長整数	302	
Form Break3	倍長整数	303	
Form Break4	倍長整数	304	
Form Break5	倍長整数	305	
Form Break6	倍長整数	306	
Form Break7	倍長整数	307	
Form Break8	倍長整数	308	
Form Break9	倍長整数	309	
Form Detail	倍長整数	0	
Form Footer	倍長整数	100	
Form Header	倍長整数	200	
Form Header1	倍長整数	201	
Form Header10	倍長整数	210	
Form Header2	倍長整数	202	
Form Header3	倍長整数	203	
Form Header4	倍長整数	204	
Form Header5	倍長整数	205	
Form Header6	倍長整数	206	
Form Header7	倍長整数	207	
Form Header8	倍長整数	208	
Form Header9	倍長整数	209	

Form Events

定数	型	値	コメント
On Activate	倍長整数	11	フォームウィンドウが最前面のウィンドウになった
On After Edit	倍長整数	45	フォーカスのあるオブジェクトの内容が更新された
On After Keystroke	倍長整数	28	フォーカスのあるオブジェクトに文字が入力されようとしている。Get edited textはこの文字を含むオブジェクトのテキストを返す
On After Sort	倍長整数	30	(リストボックスのみ) リストボックスの列中で標準のソートが行われた
On Arrow Click	倍長整数	38	(3Dボタンのみ)3D ボタンの"三角"エリアがクリックされた
On Before Data Entry	倍長整数	41	(リストボックスのみ) リストボックスセルが編集モードに変更されようとしている
On Before Keystroke	倍長整数	17	フォーカスのあるオブジェクトに文字が入力されようとしている。Get edited textはこの文字を含まないオブジェクトのテキストを返す
On Begin Drag Over	倍長整数	46	オブジェクトがドラッグされている
On Begin URL Loading	倍長整数	47	(Webエリアのみ) 新しいURLがWeb エリアにロードされた
On bound variable change	倍長整数	54	サブフォームにバインドされた変数が更新された
On Clicked	倍長整数	4	オブジェクト上でクリックされた
On Close Box	倍長整数	22	ウィンドウのクローズボックスがクリックされた
On Close Detail	倍長整数	26	入力フォームから離れ、出力フォームに移動しようとしている
On Collapse	倍長整数	44	(階層リストまたは階層リストボックスのみ) クリックやキーストロークで階層リストの要素が折りたたまれた
On Column Moved	倍長整数	32	(リストボックスのみ) リストボックスの列がユーザのドラッグ＆ドロップで移動された
On Column Resize	倍長整数	33	(リストボックスのみ) リストボックスの列幅が変更された
On Data Change	倍長整数	20	オブジェクトのデータが変更された
On Deactivate	倍長整数	12	フォームウィンドウが最前面のウィンドウでなくなった
On Delete Action	倍長整数	58	(階層リストとリストボックスのみ) ユーザーが項目の削除を試みた
On Display Detail	倍長整数	8	レコードがリスト中に、あるいは行がリストボックス中に表示されようとしている
On Double Clicked	倍長整数	13	オブジェクト上でダブルクリックされた
On Drag Over	倍長整数	21	データがオブジェクト上にドロップされる可能性がある
On Drop	倍長整数	16	データがオブジェクトにドロップされた
On End URL Loading	倍長整数	49	(Webエリアのみ) URLのすべてのリソースがロードされた
On Expand	倍長整数	43	(階層リストまたは階層リストボックスのみ) クリックやキーストロークで階層リストの要素が展開された
On Footer Click	倍長整数	57	(リストボックスのみ) リストボックスあるいはリストボックス列でフッターがクリックされた
On Getting Focus	倍長整数	15	フォームオブジェクトがフォーカスを得た
On Header	倍長整数	5	フォームのヘッダエリアが印刷あるいは表示されようとしている
On Header Click	倍長整数	42	(リストボックスのみ) リストボックスの列ヘッダでクリックが行われた
On Load	倍長整数	1	フォームが表示または印刷されようとしている
On Load Record	倍長整数	40	リスト更新中、更新中にレコードがロードされた (ユーザがレコード行をクリックし、フィールドが編集モードになった)
On Long Click	倍長整数	39	(3Dボタンのみ) 3D ボタンがクリックされ、特定の時間以上マウスボタンが押され続けている
On Losing Focus	倍長整数	14	フォームオブジェクトがフォーカスを失った
On Mac toolbar button	倍長整数	55	Mac OSにおいて、ユーザーがツールバー管理ボタンをクリックした
On Menu Selected	倍長整数	18	メニュー項目が選択された
On Mouse Enter	倍長整数	35	マウスカーソルがオブジェクトの描画エリア内に入った
On Mouse Leave	倍長整数	36	マウスカーソルがオブジェクトの描画エリアから出た
On Mouse Move	倍長整数	37	マウスカーソルがオブジェクトの描画エリア上で (最低1ピクセル) 動いたか、変更キー(Shift, Alt, Shift Lock)が押された イベントがオブジェクトに対してのみチェックされていた場合は、マウスカーソルがオブジェクトのグラフィックエリアの中にあった場合にのみイベントが生成されます。
On Open Detail	倍長整数	25	レコードがダブルクリックされて、入力フォームを開こうとしている
On Open External Link	倍長整数	52	(Webエリアのみ) 外部URLがブラウザで開かれた
On Outside Call	倍長整数	10	フォームがCALL PROCESSによる呼び出しを受けた
On Page Change	倍長整数	56	フォーム中のカレントページが変更された
On Picture Scroll	倍長整数	59	マウスやキーボードを使用して、ユーザーがピクチャーフィールドや変数の内容をスクロールした。
On Plug in Area	倍長整数	19	外部オブジェクトのオブジェクトメソッドの実行がリクエストされた
On Printing Break	倍長整数	6	フォームのブレークエリアのひとつが印刷されようとしている
On Printing Detail	倍長整数	23	フォームの詳細エリアが印刷されようとしている
On Printing Footer	倍長整数	7	フォームのフッタエリアが印刷されようとしている
On Resize	倍長整数	29	フォームウィンドウがリサイズされた
On Row Moved	倍長整数	34	(リストボックスのみ) リストボックスの行がユーザのドラッグ＆ドロップで移動された
On Selection Change	倍長整数	31	• リストボックス: 現在の行や列の選択が変更された • リスト中のレコード: リストフォームまたはサブフォームにおいて、カレントレコードあるいはカレントセレクションの行選択が変更された • 階層リスト: リスト中の選択がクリックやキーストロークなどで変更された • 入力可フィールドや変数 (v12.x以降): クリックやキー押下により、選択されたテキストやカーソルの位置がエリア内で変更された
On Timer	倍長整数	27	SET TIMERコマンドで設定した時間が経過した
On Unload	倍長整数	24	フォームを閉じる、あるいは解放しようとしている
On URL Filtering	倍長整数	51	(Webエリアのみ) Web エリアがURLをブロックした
On URL Loading Error	倍長整数	50	(Webエリアのみ) URLをロード中にエラーが発生した
On URL Resource Loading	倍長整数	48	(Webエリアのみ) 新しいリソースがWeb エリアにロードされた
On Validate	倍長整数	3	レコードのデータ入力が受け入れられた
On Window Opening Denied	倍長整数	53	(Webエリアのみ) ポップアップウィンドウがブロックされた

Form Object Types

定数	型	値	コメント
Object type 3D button	倍長整数	16	
Object type 3D checkbox	倍長整数	26	
Object type 3D radio button	倍長整数	23	
Object type button grid	倍長整数	20	
Object type checkbox	倍長整数	25	
Object type combobox	倍長整数	11	
Object type dial	倍長整数	28	
Object type group	倍長整数	21	
Object type groupbox	倍長整数	30	
Object type hierarchical list	倍長整数	6	
Object type hierarchical popup	倍長整数	13	
Object type highlight button	倍長整数	17	
Object type invisible button	倍長整数	18	
Object type line	倍長整数	32	
Object type listbox	倍長整数	7	
Object type listbox column	倍長整数	9	
Object type listbox footer	倍長整数	10	
Object type listbox header	倍長整数	8	
Object type matrix	倍長整数	35	
Object type oval	倍長整数	34	
Object type picture button	倍長整数	19	
Object type picture input	倍長整数	4	
Object type picture popup	倍長整数	14	
Object type picture radio button	倍長整数	24	
Object type plugin area	倍長整数	38	
Object type popup drop down	倍長整数	12	
Object type progress indicator	倍長整数	27	
Object type push button	倍長整数	15	
Object type radio button	倍長整数	22	
Object type radio button field	倍長整数	5	
Object type rectangle	倍長整数	31	
Object type rounded rectangle	倍長整数	33	
Object type ruler	倍長整数	29	
Object type splitter	倍長整数	36	
Object type static picture	倍長整数	2	
Object type static text	倍長整数	1	
Object type subform	倍長整数	39	
Object type tab control	倍長整数	37	
Object type text input	倍長整数	3	
Object type unknown	倍長整数	0	
Object type web area	倍長整数	40	

Form Objects (Access)

定数	型	値	コメント
Object current	倍長整数	0	
Object First in entry order	文字列	;erstesObjekt	
Object named	倍長整数	3	
Object subform container	倍長整数	2	
Object with focus	倍長整数	1	

Form Objects (Properties)

定数	型	値	コメント
Align bottom	倍長整数	4	
Align center	倍長整数	3	
Align default	倍長整数	1	
Align left	倍長整数	2	
Align right	倍長整数	4	
Align top	倍長整数	2	
Asynchronous progress bar	倍長整数	3	連続したアニメーションを表示する、回転型のインジケーター
Barber shop	倍長整数	2	連続したアニメーションを表示するバー
Border Dotted	倍長整数	2	オブジェクトの境界線は1ptの点線になります。
Border Double	倍長整数	5	オブジェクトの境界線は二重線(1ピクセル離れた2本の1ptの線)になります。
Border None	倍長整数	0	オブジェクトは境界線を持ちません。
Border Plain	倍長整数	1	オブジェクトの境界線は連続した一本の1ptの線になります。
Border Raised	倍長整数	3	オブジェクトの境界線は浮き上がったような3Dエフェクトになります。
Border Sunken	倍長整数	4	オブジェクトの境界線は沈み込んだような3Dエフェクトになります。
Border System	倍長整数	6	オブジェクトの境界線はシステムのグラフィック仕様に沿ったものになります。
Choice list	倍長整数	0	選択できる値のリスト(プロパティリスト内の「選択リスト」)。(デフォルト)
Disable events others unchanged	倍長整数	2	arrEvents に指定された全てのイベントは無効化され、他は何も変更されません。
Enable events disable others	倍長整数	0	arrEvents に指定された全てのイベントは有効になり、他は全て無効化されます。
Enable events others unchanged	倍長整数	1	arrEvents に指定された全てのイベントは有効になり、他は何も変更されません。
Excluded list	倍長整数	2	入力できない値のリスト。(プロパティリスト内の「除外リスト」)
Multiline Auto	倍長整数	0	単独行のエリアでは、行に表示しきれない単語は切り落とされ、改行はされません。 複数行のエリアでは、改行が行われます。
Multiline No	倍長整数	2	改行は禁止されます。テキストは必ず単独行として表示されます。文字列かテキストフィールドか変数に改行が含まれていたとしても、改行は行われません。
Multiline Yes	倍長整数	1	単独行のエリアでは、テキストは最初の改行までか、単語全体を表示できる最後の単語までが表示されます。その後改行が挿入されるので、↓キーを押すことによってエリアの内容をスクロールすることができます。 複数行のエリアでは、自動で改行が行われます。
Orientation 0°	倍長整数	0	回転なし(デフォルト値)
Orientation 180°	倍長整数	180	テキストの方向を時計回りに180°回転
Orientation 90° left	倍長整数	270	テキストの方向を反時計回りに90°回転
Orientation 90° right	倍長整数	90	テキストの方向を時計回りに90°回転
Print Frame fixed with multiple records	倍長整数	2	フレームは同じサイズを維持しますが、4Dは全てのレコードが載るまで複数回フォームを印刷します。
Print Frame fixed with truncation	倍長整数	1	4Dはサブフォームのエリアに収まるレコードのみ印刷します。フォームは一度だけ印刷され、印刷されなかったレコードは無視されます。
Progress bar	倍長整数	1	標準の進捗バー
Required list	倍長整数	1	入力可能な値のリスト(プロパティリストの"指定リスト"オプション)。
Resize horizontal grow	倍長整数	1	ウィンドウが横方向に拡げられたら、オブジェクトの幅も同じ比率だけ右に拡大する。
Resize horizontal move	倍長整数	2	ウィンドウの幅が拡げられたら、オブジェクトも同じだけ右方向に移動する。
Resize horizontal none	倍長整数	0	ウィンドウの幅が変更されても、オブジェクトの位置及びサイズを変更しない。
Resize vertical grow	倍長整数	1	ウィンドウが縦方向に拡げられたら、オブジェクトの高さも同じ比率だけ下に拡大する。
Resize vertical move	倍長整数	2	ウィンドウの高さが拡げられたら、オブジェクトも同じだけ下方向に移動する。
Resize vertical none	倍長整数	0	ウィンドウの高さが変更されても、オブジェクトの位置及びサイズを変更しない。

Form Parameters

定数	型	値	コメント
Multiple Selection	倍長整数	2	複数レコードを同時に選択することができます。連続しているレコードを選択するには、選択する最初のレコードをクリックし、次に**Shift**キーを押しながらセレクションに含めたい最後のレコードをクリックします。連続しないレコードを選択するには、**Ctrl**キー (Windows) または、**Command**キー (Mac OS) を押したまま各レコードをクリックします。
No Selection	倍長整数	0	リスト上のレコードを選択することはできません。
NonInverted Objects	倍長整数	0	
Single Selection	倍長整数	1	一度に1レコードだけを選択できます。

Function Keys

定数	型	値	コメント
Backspace Key	倍長整数	8	
Down Arrow Key	倍長整数	31	
End Key	倍長整数	4	
Enter Key	倍長整数	3	
Escape Key	倍長整数	27	
F1 Key	倍長整数	-122	
F10 Key	倍長整数	-109	
F11 Key	倍長整数	-103	
F12 Key	倍長整数	-111	
F13 Key	倍長整数	-105	
F14 Key	倍長整数	-107	
F15 Key	倍長整数	-113	
F2 Key	倍長整数	-120	
F3 Key	倍長整数	-99	
F4 Key	倍長整数	-118	
F5 Key	倍長整数	-96	
F6 Key	倍長整数	-97	
F7 Key	倍長整数	-98	
F8 Key	倍長整数	-100	
F9 Key	倍長整数	-101	
Help Key	倍長整数	5	
Home Key	倍長整数	1	
Left Arrow Key	倍長整数	28	
Page Down Key	倍長整数	12	
Page Up Key	倍長整数	11	
Return Key	倍長整数	13	
Right Arrow Key	倍長整数	29	
Tab Key	倍長整数	9	
Up Arrow Key	倍長整数	30	

Hierarchical Lists

定数	型	値	コメント
Additional text	文字列	4D_additional_text	
Ala Macintosh	倍長整数	1	
Ala Windows	倍長整数	2	
Macintosh node	倍長整数	860	
Use PicRef	倍長整数	131072	
Use PICT resource	倍長整数	65536	
Windows node	倍長整数	138	

HTTP Client

定数	型	値	コメント
HTTP Basic	倍長整数	1	BASIC認証メソッドを使用する
HTTP Compression	倍長整数	6	値 = 0 (圧縮しない) または 1 (圧縮する)。デフォルト値: 0 このオプションを使用して、クライアント/サーバー間通信を効率化するための圧縮メカニズムの有効/無効を切り替えることができます。このメカニズムが有効になっていると、HTTPクライアントはサーバーのレスポンスに応じてdeflateまたはgzip圧縮を使用します。
HTTP DELETE Method	文字列	DELETE	RFC 2616参照
HTTP Digest	倍長整数	2	DIGEST認証メソッドを使用する
HTTP Display auth dial	倍長整数	4	値 = 0 (ダイアログボックスを表示しない) または 1 (ダイアログボックスを表示する)。デフォルト値: 0 このオプションを使用して、HTTP GetやHTTP Requestを実行した際に認証ダイアログボックスを表示するかどうかを指定できます。デフォルトで認証ダイアログボックスは表示されず、HTTP AUTHENTICATEコマンドを使用して認証を行います。しかし認証ダイアログを表示してユーザーに認証IDを入力させたい場合、valueに1を渡します。ダイアログボックスは認証が必要な場合のみ表示されます。
HTTP Follow redirect	倍長整数	2	値 = 0 (リダイレクトを許可しない) または 1 (リダイレクトを許可する) デフォルト値 = 1
HTTP GET Method	文字列	GET	RFC 2616参照。HTTP Getを使用するのと同等。
HTTP HEAD Method	文字列	HEAD	RFC 2616参照
HTTP Max redirect	倍長整数	3	*値* = 受け入れるリダイレクト数の最大値 デフォルト値 = 2
HTTP OPTIONS Method	文字列	OPTIONS	See RFC 2616
HTTP POST Method	文字列	POST	RFC 2616参照
HTTP PUT Method	文字列	PUT	RFC 2616参照
HTTP Reset auth settings	倍長整数	5	値 = 0 (情報を削除しない) または 1 (情報を削除する)。デフォルト値: 0 このオプションを使用して4Dに、各HTTP GetやHTTP Requestコマンド実行毎に、ユーザーの認証情報 (ユーザー名、パスワード、認証メソッド) をリセットするよう指示できます。デフォルトでこれらの情報は保持され、各リクエストで再利用されます。value に1を渡すとコマンド実行毎にこれらの情報は削除されます。この設定に関わらず、これらの情報はプロセスが終了すると削除されます。
HTTP Timeout	倍長整数	1	値 = クライアントリクエストのタイムアウト (秒単位)。このタイムアウトはサーバーからのレスポンスを何秒待つかを指定します。タイムアウト時間を経過するとクライアントはセッションを閉じ、リクエストは失われます。 デフォルトでタイムアウトは20秒に設定されています。ネットワークの状態やリクエストの特性に応じてこの値を変更できます。
HTTP TRACE Method	文字列	TRACE	RFC 2616参照

Index Type

定数	型	値	コメント
Cluster BTree Index	倍長整数	3	クラスタを使用するB-Treeタイプのインデックス。このインデックスタイプは、インデックスが少数のキーを持つ場合、つまり同じ値がデータ内で頻繁に生じる場合に最も適しています。
Default Index Type	倍長整数	0	4Dはフィールドに応じて最適なインデックスのタイプを設定します (キーワードインデックスを除く) 。
Keywords Index	倍長整数	-1	フィールドの内容を単語に分割してインデックス化します。このタイプのインデックスはテキスト、文字、ピクチャー型で使用できます。警告: キーワードを複合タイプにすることはできません。
Standard BTree Index	倍長整数	1	標準 B-Treeタイプのインデックス。この多目的用のインデックスタイプは4Dの以前のバージョンで使用されています。

Is License Available

定数	型	値	コメント
4D Client SOAP License	倍長整数	808465465	
4D Client Web License	倍長整数	808465209	
4D Draw License	倍長整数	808464694	
4D for ADO License	倍長整数	808465714	
4D for MySQL License	倍長整数	808465712	
4D for OCI License	倍長整数	808465208	
4D for PostgreSQL License	倍長整数	808465713	
4D for Sybase License	倍長整数	808465715	
4D ODBC Pro License	倍長整数	808464946	
4D SOAP License	倍長整数	808465464	
4D SOAP Local License	倍長整数	808531000	
4D SOAP One Connection Licence	倍長整数	825242680	
4D SQL Server License	倍長整数	808464949	
4D SQL Server Local License	倍長整数	808530485	
4D SQL Server One Conn. Licence	倍長整数	825242165	
4D View License	倍長整数	808465207	
4D Web License	倍長整数	808464945	
4D Web Local License	倍長整数	808530481	
4D Web One Connection License	倍長整数	825242161	
4D Write License	倍長整数	808464697	

ISO Latin Character Entities

定数	型	値	コメント
ISO L1 a acute	文字列	&aacute;	
ISO L1 a circumflex	文字列	&acirc;	
ISO L1 a grave	文字列	&agrave;	
ISO L1 a ring	文字列	&aring;	
ISO L1 a tilde	文字列	&atilde;	
ISO L1 a umlaut	文字列	&auml;	
ISO L1 ae ligature	文字列	&aelig;	
ISO L1 Ampersand	文字列	&amp;	
ISO L1 c cedilla	文字列	&ccedil;	
ISO L1 Cap A acute	文字列	&Aacute;	
ISO L1 Cap A circumflex	文字列	&Acirc;	
ISO L1 Cap A grave	文字列	&Agrave;	
ISO L1 Cap A ring	文字列	&Aring;	
ISO L1 Cap A tilde	文字列	&Atilde;	
ISO L1 Cap A umlaut	文字列	&Auml;	
ISO L1 Cap AE ligature	文字列	&AElig;	
ISO L1 Cap C cedilla	文字列	&Ccedil;	
ISO L1 Cap E acute	文字列	&Eacute;	
ISO L1 Cap E circumflex	文字列	&Ecirc;	
ISO L1 Cap E grave	文字列	&Egrave;	
ISO L1 Cap E umlaut	文字列	&Euml;	
ISO L1 Cap Eth Icelandic	文字列	&ETH;	
ISO L1 Cap I acute	文字列	&Iacute;	
ISO L1 Cap I circumflex	文字列	&Icirc;	
ISO L1 Cap I grave	文字列	&Igrave;	
ISO L1 Cap I umlaut	文字列	&Iuml;	
ISO L1 Cap N tilde	文字列	&Ntilde;	
ISO L1 Cap O acute	文字列	&Oacute;	
ISO L1 Cap O circumflex	文字列	&Ocirc;	
ISO L1 Cap O grave	文字列	&Ograve;	
ISO L1 Cap O slash	文字列	&Oslash;	
ISO L1 Cap O tilde	文字列	&Otilde;	
ISO L1 Cap O umlaut	文字列	&Ouml;	
ISO L1 Cap THORN Icelandic	文字列	&THORN;	
ISO L1 Cap U acute	文字列	&Uacute;	
ISO L1 Cap U circumflex	文字列	&Ucirc;	
ISO L1 Cap U grave	文字列	&Ugrave;	
ISO L1 Cap U umlaut	文字列	&Uuml;	
ISO L1 Cap Y acute	文字列	&Yacute;	
ISO L1 Copyright	文字列	&copy;	
ISO L1 e acute	文字列	&eacute;	
ISO L1 e circumflex	文字列	&ecirc;	
ISO L1 e grave	文字列	&egrave;	
ISO L1 e umlaut	文字列	&euml;	
ISO L1 eth Icelandic	文字列	&eth;	
ISO L1 Greater than	文字列	&gt;	
ISO L1 i acute	文字列	&iacute;	
ISO L1 i circumflex	文字列	&icirc;	
ISO L1 i grave	文字列	&igrave;	
ISO L1 i umlaut	文字列	&iuml;	
ISO L1 Less than	文字列	&lt;	
ISO L1 n tilde	文字列	&ntilde;	
ISO L1 o acute	文字列	&oacute;	
ISO L1 o circumflex	文字列	&ocirc;	
ISO L1 o grave	文字列	&ograve;	
ISO L1 o slash	文字列	&oslash;	
ISO L1 o tilde	文字列	&otilde;	
ISO L1 o umlaut	文字列	&ouml;	
ISO L1 Quotation mark	文字列	&quot;	
ISO L1 Registered	文字列	&reg;	
ISO L1 sharp s German	文字列	&szlig;	
ISO L1 thorn Icelandic	文字列	&thorn;	
ISO L1 u acute	文字列	&uacute;	
ISO L1 u circumflex	文字列	&ucirc;	
ISO L1 u grave	文字列	&ugrave;	
ISO L1 u umlaut	文字列	&uuml;	
ISO L1 y acute	文字列	&yacute;	
ISO L1 y umlaut	文字列	&yuml;	

List Box

定数	型	値	コメント
Listbox add to selection	倍長整数	1	選択された行は既存の選択行に追加されます。指定した行が既存の選択に属している場合、コマンドは何も行いません。
Listbox all	倍長整数	0	コマンドはすべてのサブレベルに作用します (引数省略時のデフォルト値)。
Listbox background color	倍長整数	1	
Listbox background color array	倍長整数	1	
Listbox break row	倍長整数	2	コマンドはrowと column引数で指定された"セル"に属するサブレベルに作用します。これらの引数は標準モードのリストボックスの行および列番号を表すことに留意してください。階層表現ではありません。rowと column 引数が省略されると、コマンドは何も行いません。
Listbox display footer	倍長整数	8	0 = 非表示 1 = 表示
Listbox display header	倍長整数	0	0=非表示, 1=表示
Listbox display hor scrollbar	倍長整数	2	0=非表示, 1=表示
Listbox display ver scrollbar	倍長整数	4	0=非表示, 1=表示
Listbox font color	倍長整数	0	
Listbox font color array	倍長整数	0	
Listbox footer height	倍長整数	9	高さ (ピクセル)
Listbox header height	倍長整数	1	高さ (ピクセル)
Listbox hor scrollbar height	倍長整数	3	高さ (ピクセル)
Listbox hor scrollbar position	倍長整数	6	カーソルの位置 (ピクセル)
Listbox inherited	倍長整数	-255	
Listbox last printed row number	倍長整数	0	infoに印刷された最後の行番号を返します。これにより次に印刷される行の番号が分かります。 リストボックスに非表示行が存在したりOBJECT SET SCROLL POSITIONコマンドが呼び出されていたりすると、返される値は実際に印刷された行数よりも、大きくなることがあります。例えば行番号1, 18そして20が印刷されると、infoには20が返されます。
Listbox level	倍長整数	3	コマンドはlevel列に対応するすべてのブレーク行に作用します。この引数は標準モードのリストボックスの列番号を指定し、階層表現を考慮しません。level引数が省略されると、コマンドはなにも行いません。
Listbox lines	倍長整数	1	高さを行数で指定。4Dはフォント設定に応じて高さを計算します。
Listbox pixels	倍長整数	0	高さをピクセルで指定 (デフォルト)。
Listbox printed height	倍長整数	3	infoに実際に印刷されたオブジェクトの高さをピクセル単位で返します (ヘッダーや線等を含む)。印刷する行数がリストボックスの高さに満たない場合、高さは自動で減らされます。
Listbox printed rows	倍長整数	1	さいごのPrint object最後のコマンド呼び出し時に実際に印刷された行数をinfoに返します。この数値には階層リストボックスの場合に追加されたブレーク行も含まれます。例えばリストボックスに20行あり、奇数行が隠されている場合、infoは10になります。
Listbox printing is over	倍長整数	2	リストボックスの最後の (表示) 行が印刷されたかどうかを示すブール値をinfoに返します。True = 行は印刷された; そうでなければFalse。
Listbox remove from selection	倍長整数	2	指定された行は既存の選択行から取り除かれます。指定した行が既存の選択に属さない場合、コマンドは何も行いません。
Listbox replace selection	倍長整数	0	選択された行が新しい選択行となり、既存のものと置き換えられます。このコマンドは、ユーザが行をクリックした場合と同じ結果になります。これは (action 引数が省略された時の) デフォルトの動作です。
Listbox selection	倍長整数	1	コマンドは選択されたサブレベルに作用します。
Listbox style array	倍長整数	2	
Listbox ver scrollbar position	倍長整数	7	カーソルの位置 (ピクセル)
Listbox ver scrollbar width	倍長整数	5	幅 (ピクセル)

Listbox Footer Calculation

定数	型	値	コメント
Listbox footer average	倍長整数	6	数値または時間型の列に使用します。 結果のデフォルト型: 実数
Listbox footer count	倍長整数	5	数値、テキスト、日付、時間、ブール、またはピクチャー型の列に使用します。 結果のデフォルト型: 倍長整数
Listbox footer custom	倍長整数	1	4Dは計算を行いません。フッター変数はプログラムを使用して計算されなければなりません。 結果のデフォルト型: フッター変数に指定された型
Listbox footer max	倍長整数	3	数値、日付、時間、またはブール型の列に使用します。 結果のデフォルト型: 列配列またはフィールドに対応する型
Listbox footer min	倍長整数	2	数値、日付、時間、またはブール型の列に使用します。 結果のデフォルト型: 列配列またはフィールドに対応する型
Listbox footer std deviation	倍長整数	7	数値または時間型の列に使用します (配列タイプのリストボックスのみ)。 結果のデフォルト型: 実数
Listbox footer sum	倍長整数	4	数値、ブールまたは時間型の列に使用します。 結果のデフォルト型: 列配列またはフィールドに対応する型
Listbox footer sum squares	倍長整数	9	数値または時間型の列に使用します (配列タイプのリストボックスのみ)。 結果のデフォルト型: 実数
Listbox footer variance	倍長整数	8	数値または時間型の列に使用します (配列タイプのリストボックスのみ)。 結果のデフォルト型: 実数

Log Events

定数	型	値	コメント
Error Message	倍長整数	2	
Information Message	倍長整数	0	
Into 4D Commands Log	倍長整数	3	この値は4Dのコマンドログファイルがアクティブである場合、このファイルにmessageの内容を記録するよう4Dに指示します。 4DコマンドログファイルはSET DATABASE PARAMETERコマンド (セレクター34) を使用して有効にできます。 **注:** 4Dのログファイルは、**Logs**フォルダに配置されます。このフォルダーはデータベースのストラクチャーファイルと同階層に作成されます(Get 4D folderコマンドを参照) 。
Into 4D Debug Message	倍長整数	1	この値は4Dにmessageをシステムデバッギング環境へ送るよう指示します。結果はプラットホームにより異なります。 • Mac OSでは、コマンドはメッセージをコンソールへ送ります。 • Windowsでは、コマンドはメッセージをデバッグメッセージとして送ります。このメッセージを読むには、Microsoft Visual StudioまたはDebugViewユーティリティが必要です。(http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx)
Into 4D Diagnostic Log	倍長整数	5	ログファイルが有効である場合に、メッセージを4Dのログファイルに記録するよう指示します。 ログファイルはSET DATABASE PARAMETERコマンド (セレクター79) を使用して有効にできます。
Into 4D Request Log	倍長整数	2	この値は4Dリクエストログがアクティブである場合、このファイルにmessageを記録するよう4Dに指示します。
Into Windows Log Events	倍長整数	0	この値は、4Dにmessage をWindowsの"Log events" へ送るよう指示します。このログは起動しているアプリケーションから送られるメッセージを受け取り保存します。この場合オプションのimportance引数を使用してmessageの重要度を設定できます (後述)。 **Notes:** • この特性を利用するには、Windows Log Eventsサービスが起動していなければなりません。 • Mac OSでは、コマンドはこの出力タイプでは何もしません。
Warning Message	倍長整数	1	

Math

定数	型	値	コメント
Degree	実数	0.0174532925199432958	
e number	実数	2.71828182845904524	
Pi	実数	3.141592653589793239	
Radian	実数	57.29577951308232088	

Menu Item Properties

定数	型	値	コメント
Access Privileges	文字列	4D_access_group	コマンドにアクセスグループを設定するために使用します。 0 = All Groups >0 = Group ID
Associated Standard Action	文字列	4D_standard_action	メニュー項目に標準アクションを割り当てるために使用します。 "Value for Associated Standard Action" テーマの定数参照。
Start a New Process	文字列	4D_start_new_process	"新規プロセス開始オプションを有効にします。 0 = No, 1 = Yes

Multistyle Text

定数	型	値	コメント
ST 4D Expressions as sources	倍長整数	2	4D式参照のオリジナルの文字列が返されます。
ST 4D Expressions as values	倍長整数	1	4D 式参照は評価された形で返されます(フォーム内のデフォルトの動作)。
ST End highlight	倍長整数	-1001	オブジェクト内のカレントのテキストセレクションの最後の文字を指定します(*)
ST End text	倍長整数	0	オブジェクトに含まれるテキストの最後の文字を指定します。
ST Expression type	倍長整数	2	セレクションには式参照のみ含まれます。
ST Expressions display mode	倍長整数	1	value 引数には以下のどちらかが入ります: • ST Values: 式の計算された値を表示します(デフォルト) • ST References: 式のソースの文字列を表示します
ST Mixed type	倍長整数	3	セレクションには少なくとも二つの異なる型のコンテンツが含まれます。
ST Plain type	倍長整数	0	セレクションにはテキストのみ含まれ、参照はありません。
ST References	倍長整数	1	式のソースの文字列を表示します。
ST References as spaces	倍長整数	0	それぞれの参照は、ノンブレークスペース文字として返されます(他のコマンドで使用されるデフォルトの動作)
ST Start highlight	倍長整数	-1000	オブジェクト内のテキストのカレントセレクションの最初の文字をを指定します (*)
ST Start text	倍長整数	1	オブジェクト内に含まれるテキストの最初の文字を指定します。
ST Tags as plain text	倍長整数	64	タグのラベルは標準テキストとして返されます。例えば、'<img src="test.jpg" alt="picture">my picture</img>'というタグの標準テキストは "my picture" となります(フォームでのデフォルトの動作)。
ST Tags as XML code	倍長整数	128	XMLのコードは標準テキストとして返されます。例えば、'<img src="test.jpg" alt="picture">my picture</img>'というタグの標準テキストは '<img src="test.jpg" alt="picture">my picture</img>' となります。
ST Text displayed with 4D Expression sources	倍長整数	86	テキストは、4D式のオリジナルの文字列とともに、表示されたままの状態が返されます。 既定済みの定数の組み合わせの 2+4+16+64 に対応します。
ST Text displayed with 4D Expression values	倍長整数	85	テキストは、4D式が評価された形で、フォームで表示されたままの形で返されます。 既定済みの定数の組み合わせの 1+4+16+64 に対応します。
ST Unknown tag type	倍長整数	4	セレクションには、未知の型のタグのみ含まれます。
ST URL as labels	倍長整数	4	「こちらのサイトまでどうぞ」といったようなURLの表示ラベルが返されます(フォームのデフォルトの機能)
ST URL as links	倍長整数	8	"http://www.4d.com" のようにリンクが返されます。
ST Url type	倍長整数	1	セレクションにはURL参照のみ含まれます。
ST User links as labels	倍長整数	16	ユーザーリンクの表示ラベルが返されます(フォームのデフォルトの機能)
ST User links as links	倍長整数	32	ユーザーリンクの中身が返されます。
ST User type	倍長整数	5	セレクションには、カスタムの参照のみ含まれます。
ST Values	倍長整数	0	4D式の、計算された値を表示します。

Multistyle Text Attributes

定数	型	値	コメント
Attribute background color	倍長整数	8	*attValue* = (Windowsのみ) 16進値またはHTMLカラー名
Attribute bold style	倍長整数	1	*attValue* = 0: 選択部からボールド属性を取り除きます *attValue* = 1: 選択部にボールド属性を適用します
Attribute font name	倍長整数	5	*attValue* = フォントファミリー名 (文字)
Attribute italic style	倍長整数	2	*attValue* = 0: 選択部からイタリック属性を取り除きます *attValue* = 1: 選択部にイタリック属性を適用します
Attribute strikethrough style	倍長整数	3	*attValue* = 0: 選択部から取り消し線属性を取り除きます *attValue* = 1: 選択部に取り消し線属性を適用します
Attribute text color	倍長整数	7	*attValue* = 16進値またはHTMLカラー名
Attribute text size	倍長整数	6	*attValue* = ポイント数 (数値)
Attribute underline style	倍長整数	4	*attValue* = 0: 選択部から下線属性を取り除きます *attValue* = 1: 選択部に下線属性を適用します

Open Form Window

定数	型	値	コメント
_O_Compositing mode form	倍長整数	4096	*** 廃止予定の定数 ***
At the Bottom	倍長整数	393216	
At the Top	倍長整数	327680	
Has full screen mode Mac	倍長整数	65536	
Has toolbar button Mac OS	倍長整数	8192	
Horizontally Centered	倍長整数	65536	
Modal form dialog box	倍長整数	1	
Movable form dialog box	倍長整数	5	
On the Left	倍長整数	131072	
On the Right	倍長整数	196608	
Palette form window	倍長整数	1984	
Plain form window	倍長整数	8	
Pop up form window	倍長整数	32	
Sheet form window	倍長整数	33	
Vertically Centered	倍長整数	262144	

Open window

定数	型	値	コメント
_O_Compositing Mode	倍長整数	4096	*** 廃止予定の定数 ***
Alternate dialog box	倍長整数	3	フローティング可
Has full screen mode Mac	倍長整数	65536	
Has grow box	倍長整数	4	
Has highlight	倍長整数	1	
Has toolbar button Mac OS	倍長整数	8192	
Has window title	倍長整数	2	
Has zoom box	倍長整数	8	
Modal dialog box	倍長整数	1	
Movable dialog box	倍長整数	5	フローティング可
Palette window	倍長整数	1984	フローティング可
Plain dialog box	倍長整数	2	フローティング可
Plain fixed size window	倍長整数	4	
Plain no zoom box window	倍長整数	0	
Plain window	倍長整数	8	
Pop up window	倍長整数	32	
Resizable sheet window	倍長整数	34	
Round corner window	倍長整数	16	
Sheet window	倍長整数	33	
Texture appearance	倍長整数	2048	

Pasteboard

定数	型	値	コメント
No such data in pasteboard	倍長整数	-102	
Picture data	文字列	PICT	
Text data	文字列	TEXT	

PHP

定数	型	値	コメント
PHP Privileges	倍長整数	1	スクリプト実行に関連して指定されるユーザ権限定義 **とりうる値**: 以下の形式の文字列 "User:Password"。例: "root:jd51254d"
PHP Raw result	倍長整数	2	結果がテキスト型のときに実行結果中にPHPから返されるHTTPヘッダに関する処理モードの定義 (結果がBLOB型のときヘッダは常に保持されます)。 **とりうる値**: ブール。False (デフォルト値) = HTTPヘッダを結果から取り除く、True = HTTPヘッダを保持する

Picture Compression

定数	型	値	コメント
QT Animation compressor	文字列	rle	
QT Compact video compressor	文字列	cdvc	
QT Graphics compressor	文字列	smc	
QT Photo compressor	文字列	jpeg	
QT Raw compressor	文字列	raw	
QT Video compressor	文字列	rpza	

Picture Display Formats

定数	型	値	コメント
On Background	倍長整数	3	
Replicated	倍長整数	7	
Scaled to Fit	倍長整数	2	
Scaled to fit prop centered	倍長整数	6	
Scaled to fit proportional	倍長整数	5	
Truncated Centered	倍長整数	1	
Truncated non Centered	倍長整数	4	

Picture Metadata Names

定数	型	値	コメント
EXIF Aperture Value	文字列	EXIF/ApertureValue	とりうる値: 実数 (APEX 値)
EXIF Brightness Value	文字列	EXIF/BrightnessValue	とりうる値: 実数 (APEX 値)
EXIF Color Space	文字列	EXIF/ColorSpace	とりうる値: EXIF Adobe RGB, EXIF s RGB, EXIF Uncalibrated
EXIF Components Configuration	文字列	EXIF/ComponentsConfiguration	とりうる値: EXIF B, EXIF Cb, EXIF Cr, EXIF G, EXIF R, EXIF Unused, EXIF Y
EXIF Compressed Bits Per Pixel	文字列	EXIF/CompressedBitsPerPixel	とりうる値: 実数
EXIF Contrast	文字列	EXIF/Contrast	とりうる値: EXIF High, EXIF Low, EXIF Normal
EXIF Custom Rendered	文字列	EXIF/CustomRendered	とりうる値: EXIF Normal, EXIF Custom
EXIF Date Time Digitized	文字列	EXIF/DateTimeDigitized	とりうる値: 日付またはテキスト (XML Datetime)
EXIF Date Time Original	文字列	EXIF/DateTimeOriginal	とりうる値: 日付またはテキスト (XML Datetime)
EXIF Digital Zoom Ratio	文字列	EXIF/DigitalZoomRatio	とりうる値: 実数
EXIF Exif Version	文字列	EXIF/ExifVersion	とりうる値: 整数配列 (4要素)
EXIF Exposure Bias Value	文字列	EXIF/ExposureBiasValue	とりうる値: 実数
EXIF Exposure Index	文字列	EXIF/ExposureIndex	とりうる値: 実数
EXIF Exposure Mode	文字列	EXIF/ExposureModus	とりうる値: EXIF Auto, EXIF Auto Bracket, EXIF Manual
EXIF Exposure Program	文字列	EXIF/ExposureProgram	とりうる値: EXIF Manual, EXIF Action, EXIF Aperture Priority AE, EXIF Creative, EXIF Landscape, EXIF Exposure Portrait, EXIF Program AE, EXIF Shutter Speed Priority AE
EXIF Exposure Time	文字列	EXIF/ExposureTime	とりうる値: 実数
EXIF F Number	文字列	EXIF/FNumber	とりうる値: 実数
EXIF File Source	文字列	EXIF/FileSource	とりうる値: EXIF Digital Camera, EXIF Film Scanner, EXIF Reflection Print Scanner
EXIF Flash	文字列	EXIF/Flash	とりうる値: EXIF Auto Mode, EXIF Compulsory Flash Firing, EXIF Compulsory Flash Suppression, EXIF Unknown, EXIF Detected, EXIF No Detection Function, EXIF Not Detected, EXIF Reserved
EXIF Flash Energy	文字列	EXIF/FlashEnergy	とりうる値: 実数
EXIF Flash Fired	文字列	EXIF/Flash/Fired	とりうる値: ブール
EXIF Flash Function Present	文字列	EXIF/Flash/FunctionPresent	とりうる値: ブール
EXIF Flash Mode	文字列	EXIF/Flash/Mode	とりうる値: EXIF Auto Mode, EXIF Compulsory Flash Firing, EXIF Compulsory Flash Suppression, EXIF Unknown
EXIF Flash Pix Version	文字列	EXIF/FlashPixVersion	とりうる値: 整数配列 (4要素)
EXIF Flash Red Eye Reduction	文字列	EXIF/Flash/RedEyeReduction	とりうる値: ブール
EXIF Flash Return Light	文字列	EXIF/Flash/ReturnLight	とりうる値: EXIF Detected, EXIF No Detection Function, EXIF Not Detected, EXIF Reserved
EXIF Focal Len In 35 mm Film	文字列	EXIF/FocalLenIn35mmFilm	とりうる値: 倍長整数
EXIF Focal Length	文字列	EXIF/FocalLength	とりうる値: 実数
EXIF Focal Plane Resolution Unit	文字列	EXIF/FocalPlaneResolutionUnit	とりうる値: 倍長整数
EXIF Focal Plane X Resolution	文字列	EXIF/FocalPlaneXResolution	とりうる値: 実数
EXIF Focal Plane Y Resolution	文字列	EXIF/FocalPlaneYResolution	とりうる値: 実数
EXIF Gain Control	文字列	EXIF/GainControl	とりうる値: EXIF High Gain Down, EXIF High Gain Up, EXIF Low Gain Down, EXIF Low Gain Up, EXIF None
EXIF Gamma	文字列	EXIF/Gamma	とりうる値: 実数
EXIF Image Unique ID	文字列	EXIF/ImageUniqueID	とりうる値: テキスト
EXIF ISO Speed Ratings	文字列	EXIF/ISOSpeedRatings	とりうる値: 倍長整数または倍長整数配列
EXIF Light Source	文字列	EXIF/LightSource	とりうる値: EXIF Unknown, EXIF Cloudy, EXIF Cool White Fluorescent, EXIF D50, EXIF D55, EXIF D65, EXIF D75, EXIF Daylight, EXIF Daylight Fluorescent, EXIF Day White Fluorescent, EXIF Fine Weather, EXIF Flash, EXIF Light Fluorescent, EXIF ISOStudio Tungsten, EXIF Other, EXIF Shade, EXIF Standard Light A, EXIF Standard Light B, EXIF Standard Light C, EXIF Tungsten, EXIF White Fluorescent
EXIF Maker Note	文字列	EXIF/MakerNote	とりうる値: テキスト
EXIF Max Aperture Value	文字列	EXIF/MaxApertureValue	とりうる値: 実数
EXIF Metering Mode	文字列	EXIF/MeteringMode	とりうる値: EXIF Other, EXIF Average, EXIF Center Weighted Average, EXIF Multi Segment, EXIF Multi Spot, EXIF Partial, EXIF Spot
EXIF Pixel X Dimension	文字列	EXIF/PixelXDimension	とりうる値: 倍長整数
EXIF Pixel Y Dimension	文字列	EXIF/PixelYDimension	とりうる値: 倍長整数
EXIF Related Sound File	文字列	EXIF/RelatedSoundFile	とりうる値: テキスト
EXIF Saturation	文字列	EXIF/Saturation	とりうる値: EXIF High, EXIF Low, EXIF Normal
EXIF Scene Capture Type	文字列	EXIF/SceneCaptureType	とりうる値: EXIF Scene Landscape, EXIF Night, EXIF Scene Portrait, EXIF Standard
EXIF Scene Type	文字列	EXIF/SceneType	とりうる値: 倍長整数
EXIF Sensing Method	文字列	EXIF/SensingMethod	とりうる値: EXIF Color Sequential Area, EXIF Color Sequential Linear, EXIF Not Defined, EXIF One Chip Color Area, EXIF Three Chip Color Area, EXIF Trilinear, EXIF Two Chip Color Area
EXIF Sharpness	文字列	EXIF/Sharpness	とりうる値: EXIF High, EXIF Low, EXIF Normal
EXIF Shutter Speed Value	文字列	EXIF/ShutterSpeedValue	とりうる値: 実数
EXIF Spectral Sensitivity	文字列	EXIF/SpectralSensitivity	とりうる値: テキスト
EXIF Subject Area	文字列	EXIF/SubjectArea	とりうる値: 倍長整数配列 (2, 3, または4要素)
EXIF Subject Dist Range	文字列	EXIF/SubjectDistRange	とりうる値: EXIF Unknown, EXIF Close, EXIF Distant, EXIF Macro
EXIF Subject Distance	文字列	EXIF/SubjectDistance	とりうる値: 実数
EXIF Subject Location	文字列	EXIF/SubjectLocation	とりうる値: 倍長整数配列 (2要素)
EXIF User Comment	文字列	EXIF/UserComment	とりうる値: テキスト
EXIF White Balance	文字列	EXIF/WhiteBalance	とりうる値: EXIF Auto, EXIF Manual
GPS Altitude	文字列	GPS/Altitude	とりうる値: GPS Above Sea Level, GPS Below Sea Level
GPS Altitude Ref	文字列	GPS/AltitudeRef	とりうる値: GPS Above Sea Level, GPS Below Sea Level
GPS Area Information	文字列	GPS/AreaInformation	とりうる値: テキスト
GPS Date Time	文字列	GPS/DateTime	とりうる値: 日付またはテキスト (XML Datetime)
GPS Dest Bearing	文字列	GPS/DestBearing	とりうる値: テキスト (1文字)
GPS Dest Bearing Ref	文字列	GPS/DestBearingRef	とりうる値: テキスト (1文字)
GPS Dest Distance	文字列	GPS/DestDistance	とりうる値: テキスト (1文字)
GPS Dest Distance Ref	文字列	GPS/DestDistanceRef	とりうる値: テキスト (1文字)
GPS Dest Latitude	文字列	GPS/DestLatitude	とりうる値: テキスト
GPS Dest Latitude Deg	文字列	GPS/DestLatitude/Deg	とりうる値: 実数
GPS Dest Latitude Dir	文字列	GPS/DestLatitude/Dir	とりうる値: テキスト (1文字)
GPS Dest Latitude Min	文字列	GPS/DestLatitude/Min	とりうる値: 実数
GPS Dest Latitude Sec	文字列	GPS/DestLatitude/Sec	とりうる値: 実数
GPS Dest Longitude	文字列	GPS/DestLongitude	とりうる値: テキスト
GPS Dest Longitude Deg	文字列	GPS/DestLongitude/Deg	とりうる値: 実数
GPS Dest Longitude Dir	文字列	GPS/DestLongitude/Dir	とりうる値: テキスト (1文字)
GPS Dest Longitude Min	文字列	GPS/DestLongitude/Min	とりうる値: 実数
GPS Dest Longitude Sec	文字列	GPS/DestLongitude/Sec	とりうる値: 実数
GPS Differential	文字列	GPS/Differential	とりうる値: GPS Correction Applied, GPS Correction Not Applied
GPS DOP	文字列	GPS/DOP	とりうる値: 実数
GPS Img Direction	文字列	GPS/ImgDirection	とりうる値: GPS Magnetic north, GPS True north
GPS Img Direction Ref	文字列	GPS/ImgDirectionRef	とりうる値: GPS Magnetic north, GPS True north
GPS Latitude	文字列	GPS/Latitude	とりうる値: GPS North, GPS South
GPS Latitude Deg	文字列	GPS/Latitude/Deg	とりうる値: 実数
GPS Latitude Dir	文字列	GPS/Latitude/Dir	とりうる値: GPS North, GPS South
GPS Latitude Min	文字列	GPS/Latitude/Min	とりうる値: 実数
GPS Latitude Sec	文字列	GPS/Latitude/Sec	とりうる値: 実数
GPS Longitude	文字列	GPS/Longitude	とりうる値: GPS West, GPS East
GPS Longitude Deg	文字列	GPS/Longitude/Deg	とりうる値: 実数
GPS Longitude Dir	文字列	GPS/Longitude/Dir	とりうる値: GPS West, GPS East
GPS Longitude Min	文字列	GPS/Longitude/Min	とりうる値: 実数
GPS Longitude Sec	文字列	GPS/Longitude/Sec	とりうる値: 実数
GPS Map Date	文字列	GPS/MapDate	とりうる値: テキスト
GPS Measure Mode	文字列	GPS/MeasureMode	とりうる値: GPS 2D, GPS 3D
GPS Processing Method	文字列	GPS/ProcessingMethod	とりうる値: テキスト
GPS Satellites	文字列	GPS/Satellites	とりうる値: テキスト
GPS Speed	文字列	GPS/Speed	とりうる値: GPS km h, GPS miles h, GPS knots h
GPS Speed Ref	文字列	GPS/SpeedRef	とりうる値: GPS km h, GPS miles h, GPS knots h
GPS Status	文字列	GPS/Status	とりうる値: GPS Measurement in progress, GPS Measurement Interoperability
GPS Track	文字列	GPS/Track	とりうる値: 実数 (0.00..359.99)
GPS Track Ref	文字列	GPS/TrackRef	とりうる値: テキスト (1文字)
GPS Version ID	文字列	GPS/VersionID	とりうる値: 倍長整数配列 (4文字)
IPTC Byline	文字列	IPTC/Byline	とりうる値: テキストまたはテキスト配列
IPTC Byline Title	文字列	IPTC/BylineTitle	とりうる値: テキストまたはテキスト配列
IPTC Caption Abstract	文字列	IPTC/CaptionAbstract	とりうる値: テキスト
IPTC Category	文字列	IPTC/Category	とりうる値: テキスト
IPTC City	文字列	IPTC/City	とりうる値: テキスト
IPTC Contact	文字列	IPTC/Contact	とりうる値: テキストまたはテキスト配列
IPTC Content Location Code	文字列	IPTC/ContentLocationCode	とりうる値: テキストまたはテキスト配列
IPTC Content Location Name	文字列	IPTC/ContentLocationName	とりうる値: テキストまたはテキスト配列
IPTC Copyright Notice	文字列	IPTC/CopyrightNotice	とりうる値: テキスト
IPTC Country Primary Location Code	文字列	IPTC/CountryPrimaryLocationCode	とりうる値: テキスト
IPTC Country Primary Location Name	文字列	IPTC/CountryPrimaryLocationName	とりうる値: テキスト
IPTC Credit	文字列	IPTC/Credit	とりうる値: テキスト
IPTC Date Time Created	文字列	IPTC/DateTimeCreated	とりうる値: 日付またはテキスト (XML Datetime)
IPTC Digital Creation Date Time	文字列	IPTC/DigitalCreationDateTime	とりうる値: 日付またはテキスト (XML Datetime)
IPTC Edit Status	文字列	IPTC/EditStatus	とりうる値: テキスト
IPTC Expiration Date Time	文字列	IPTC/ExpirationDateTime	とりうる値: 日付またはテキスト (XML Datetime)
IPTC Fixture Identifier	文字列	IPTC/FixtureIdentifier	とりうる値: テキスト
IPTC Headline	文字列	IPTC/Headline	とりうる値: テキスト
IPTC Image Orientation	文字列	IPTC/ImageOrientation	とりうる値: テキスト
IPTC Image Type	文字列	IPTC/ImageType	とりうる値: テキスト
IPTC Keywords	文字列	IPTC/Keywords	とりうる値: テキストまたはテキスト配列
IPTC Language Identifier	文字列	IPTC/LanguageIdentifier	とりうる値: テキスト
IPTC Object Attribute Reference	文字列	IPTC/ObjectAttributeReference	とりうる値: テキスト
IPTC Object Cycle	文字列	IPTC/ObjectCycle	とりうる値: テキスト
IPTC Object Name	文字列	IPTC/ObjektName	とりうる値: テキスト
IPTC Original Transmission Reference	文字列	IPTC/OriginalTransmissionReference	とりうる値: テキスト
IPTC Originating Program	文字列	IPTC/OriginatingProgram	とりうる値: テキスト
IPTC Program	文字	IPTC/ProgramVersion	とりうる値: テキスト

Version	列		
IPTC Province State	文字列	IPTC/ProvinceState	**とりうる値:** テキスト
IPTC Release Date Time	文字列	IPTC/ReleaseDateTime	**とりうる値:** 日付またはテキスト (XML Datetime)
IPTC Scene	文字列	IPTC/Scene	**とりうる値:** IPTC Action, IPTC Aerial View, IPTC Close Up, IPTC Couple, IPTC Exterior View, IPTC Full Length, IPTC General View, IPTC Group, IPTC Half Length, IPTC Headshot, IPTC Interior View, IPTC Movie Scene, IPTC Night Scene, IPTC Off Beat, IPTC Panoramic View, IPTC Performing, IPTC Posing, IPTC Profile, IPTC Rear View, IPTC Satellite, IPTC Single, IPTC Symbolic, IPTC Two, IPTC Under Water
IPTC Source	文字列	IPTC/Source	**とりうる値:** テキスト
IPTC Special Instructions	文字列	IPTC/SpecialInstructions	**とりうる値:** テキスト
IPTC Star Rating	文字列	IPTC/StarRating	**とりうる値:** 倍長整数
IPTC Sub Location	文字列	IPTC/SubLocation	**とりうる値:** テキスト
IPTC Subject Reference	文字列	IPTC/SubjectReference	**とりうる値:** 倍長整数または倍長整数配列
IPTC Supplemental Category	文字列	IPTC/SupplementalCategory	**とりうる値:** テキストまたはテキスト配列
IPTC Urgency	文字列	IPTC/Urgency	**とりうる値:** 倍長整数
IPTC Writer Editor	文字列	IPTC/WriterEditor	**とりうる値:** テキストまたはテキスト配列
TIFF Artist	文字列	TIFF/Artist	**とりうる値:** テキスト
TIFF Compression	文字列	TIFF/Compression	**とりうる値:** TIFF Adobe Deflate, TIFF CCIRLEW, TIFF CCITT1D, TIFF DCS, TIFF Deflate, TIFF Epson ERF, TIFF IT8BL, TIFF IT8CTPAD, TIFF IT8LW, TIFF IT8MP, TIFF JBIG, TIFF JBIGB&W, TIFF JBIGColor, TIFF JPEG, TIFF JPEG2000, TIFF JPEGThumbs Only, TIFF Kodak262, TIFF Kodak DCR, TIFF Kodak KDC, TIFF LZW, TIFF MDIBinary Level Codec, TIFF MDIProgressive Transform Codec, TIFF MDIVector, TIFF Next, TIFF Nikon NEF, TIFF Pack Bits, TIFF Pentax PEF, TIFF Pixar Film, TIFF Pixar Log, TIFF SGILog, TIFF SGILog24, TIFF Sony ARW, TIFF T4Group3Fax, TIFF T6Group4Fax, TIFF Thunderscan, TIFF Uncompressed
TIFF Copyright	文字列	TIFF/Copyright	**とりうる値:** テキスト
TIFF Date Time	文字列	TIFF/DateTime	**とりうる値:** 日付またはテキスト (XML Datetime)
TIFF Document Name	文字列	TIFF/DocumentName	**とりうる値:** テキスト
TIFF Host Computer	文字列	TIFF/HostComputer	**とりうる値:** テキスト
TIFF Image Description	文字列	TIFF/ImageDescription	**とりうる値:** テキスト
TIFF Make	文字列	TIFF/Make	**とりうる値:** テキスト
TIFF Model	文字列	TIFF/Model	**とりうる値:** テキスト
TIFF Orientation	文字列	TIFF/Orientation	**とりうる値:** TIFF Horizontal, TIFF Mirror Horizontal, TIFF Mirror Horizontal And Rotate270CW, TIFF Mirror Horizontal And Rotate90CW, TIFF Mirror Vertical, TIFF Rotate180, TIFF Rotate270CW, TIFF Rotate90CW
TIFF Photometric Interpretation	文字列	TIFF/PhotometricInterpretation	**とりうる値:** TIFF Black Is Zero, TIFF CIELab, TIFF CMYK, TIFF Color Filter Array, TIFF ICCLab, TIFF ITULab, TIFF Linear Raw, TIFF Pixar Log L, TIFF Pixar Log Luv, TIFF RGB, TIFF RGBPalette, TIFF Transparency Mask, TIFF White Is Zero, TIFF YCb Cr
TIFF Resolution Unit	文字列	TIFF/ResolutionUnit	**とりうる値:** TIFF CM, TIFF Inches, TIFF MM, TIFF None, TIFF UM
TIFF Software	文字列	TIFF/Software	**とりうる値:** テキスト
TIFF XResolution	文字列	TIFF/XResolution	**とりうる値:** 実数
TIFF YResolution	文字列	TIFF/YResolution	**とりうる値:** 実数

Picture Metadata Values

定数	型	値	コメント
EXIF Action	倍長整数	6	
EXIF Adobe RGB	倍長整数	2	
EXIF Aperture Priority AE	倍長整数	3	
EXIF Auto	倍長整数	0	
EXIF Auto Bracket	倍長整数	2	
EXIF Auto Mode	倍長整数	3	
EXIF Average	倍長整数	1	
EXIF B	倍長整数	6	
EXIF Cb	倍長整数	2	
EXIF Center Weighted Average	倍長整数	2	
EXIF Close	倍長整数	2	
EXIF Cloudy	倍長整数	10	
EXIF Color Sequential Area	倍長整数	5	
EXIF Color Sequential Linear	倍長整数	8	
EXIF Compulsory Flash Firing	倍長整数	1	
EXIF Compulsory Flash Suppression	倍長整数	2	
EXIF Cool White Fluorescent	倍長整数	14	
EXIF Cr	倍長整数	3	
EXIF Creative	倍長整数	5	
EXIF Custom	倍長整数	1	
EXIF D50	倍長整数	23	
EXIF D55	倍長整数	20	
EXIF D65	倍長整数	21	
EXIF D75	倍長整数	22	
EXIF Day White Fluorescent	倍長整数	13	
EXIF Daylight	倍長整数	1	
EXIF Daylight Fluorescent	倍長整数	12	
EXIF Detected	倍長整数	3	
EXIF Digital Camera	倍長整数	3	
EXIF Distant	倍長整数	3	
EXIF Exposure Portrait	倍長整数	7	
EXIF Film Scanner	倍長整数	1	
EXIF Fine Weather	倍長整数	9	
EXIF Flashlight	倍長整数	4	
EXIF G	倍長整数	5	
EXIF High	倍長整数	2	
EXIF High Gain Down	倍長整数	4	
EXIF High Gain Up	倍長整数	2	
EXIF ISOStudio Tungsten	倍長整数	24	
EXIF Landscape	倍長整数	8	
EXIF Light Fluorescent	倍長整数	2	
EXIF Low	倍長整数	1	
EXIF Low Gain Down	倍長整数	3	
EXIF Low Gain Up	倍長整数	1	
EXIF Macro	倍長整数	1	
EXIF Manual	倍長整数	1	
EXIF Multi Segment	倍長整数	5	
EXIF Multi Spot	倍長整数	4	
EXIF Night	倍長整数	3	
EXIF No Detection Function	倍長整数	0	
EXIF None	倍長整数	0	
EXIF Normal	倍長整数	0	
EXIF Not Defined	倍長整数	1	
EXIF Not Detected	倍長整数	2	
EXIF One Chip Color Area	倍長整数	2	
EXIF Other	倍長整数	255	
EXIF Partial	倍長整数	6	
EXIF Program AE	倍長整数	2	
EXIF R	倍長整数	4	
EXIF Reflection Print Scanner	倍長整数	2	
EXIF Reserved	倍長整数	1	
EXIF s RGB	倍長整数	1	
EXIF Scene Landscape	倍長整数	1	
EXIF Scene Portrait	倍長整数	2	
EXIF Shade	倍長整数	11	
EXIF Shutter Speed Priority AE	倍長整数	4	
EXIF Spot	倍長整数	3	
EXIF Standard	倍長整数	0	
EXIF Standard Light A	倍長整数	17	
EXIF Standard Light B	倍長整数	18	
EXIF Standard Light C	倍長整数	19	
EXIF Three Chip Color Area	倍長整数	4	
EXIF Trilinear	倍長整数	7	
EXIF Tungsten	倍長整数	3	
EXIF Two Chip Color Area	倍長整数	3	
EXIF Uncalibrated	倍長整数	-1	
EXIF Unknown	倍長整数	0	
EXIF Unused	倍長整数	0	
EXIF White Fluorescent	倍長整数	15	
EXIF Y	倍長整数	1	
GPS 2D	倍長整数	2	
GPS 3D	倍長整数	3	
GPS Above Sea Level	倍長整数	0	
GPS Below Sea Level	倍長整数	1	
GPS Correction Applied	倍長整数	1	
GPS Correction Not Applied	倍長整数	0	
GPS East	文字列	E	
GPS km h	文字列	K	
GPS knots h	文字列	K	
GPS Magnetic north	文字列	M	
GPS Measurement in progress	文字列	A	
GPS Measurement Interoperability	文字列	V	
GPS miles h	文字列	M	
GPS North	文字列	N	
GPS South	文字列	S	
GPS True north	文字列	T	
GPS West	文字列	W	
IPTC Action	倍長整数	11900	
IPTC Aerial View	倍長整数	11200	
IPTC Close Up	倍長整数	11800	
IPTC Couple	倍長整数	10700	
IPTC Exterior View	倍長整数	11600	
IPTC Full Length	倍長整数	10300	
IPTC General View	倍長整数	11000	
IPTC Group	倍長整数	10900	
IPTC Half Length	倍長整数	10200	
IPTC Headshot	倍長整数	10100	
IPTC Interior View	倍長整数	11700	
IPTC Movie Scene	倍長整数	12400	
IPTC Night Scene	倍長整数	11400	
IPTC Off Beat	倍長整数	12300	
IPTC Panoramic View	倍長整数	11100	
IPTC Performing	倍長整数	12000	
IPTC Posing	倍長整数	12100	
IPTC Profile	倍長整数	10400	
IPTC Rear View	倍長整数	10500	
IPTC Satellite	倍長整数	11500	
IPTC Single	倍長整数	10600	
IPTC Symbolic	倍長整数	12200	
IPTC Two	倍長整数	10800	
IPTC Under Water	倍長整数	11300	
TIFF Adobe Deflate	倍長整数	8	
TIFF Black Is Zero	倍長整数	1	
TIFF CCIRLEW	倍長整数	32771	
TIFF CCITT1D	倍長整数	2	
TIFF CIELab	倍長整数	8	
TIFF CM	倍長整数	3	
TIFF CMYK	倍長整数	5	
TIFF Color Filter Array	倍長整数	32803	
TIFF DCS	倍長整数	32947	
TIFF Deflate	倍長整数	32946	
TIFF Epson ERF	倍長整数	32769	
TIFF Horizontal	倍長整数	1	
TIFF ICCLab	倍長整数	9	
TIFF Inches	倍長整数	2	
TIFF IT8BL	倍長整数	32898	
TIFF IT8CTPAD	倍長整数	32895	
TIFF IT8LW	倍長整数	32896	
TIFF IT8MP	倍長整数	32897	
TIFF ITULab	倍長整数	10	
TIFF JBIG	倍長整数	34661	
TIFF JBIGB&W	倍長整数	9	
TIFF JBIGColor	倍長整数	10	
TIFF JPEG	倍長整数	7	
TIFF JPEG2000	倍長整数	34712	
TIFF JPEGThumbs Only	倍長整数	6	
TIFF Kodak DCR	倍長整数	65000	
TIFF Kodak KDC	倍長整数	32867	
TIFF Kodak262	倍長整数	262	
TIFF Linear Raw	倍長整数	34892	
TIFF LZW	倍長整数	5	
TIFF MDIBinary Level Codec	倍長整数	34718	
TIFF MDIProgressive Transform Codec	倍長整数	34719	
TIFF MDIVector	倍長整数	34720	
TIFF Mirror Horizontal	倍長整数	2	
TIFF Mirror Horizontal And Rotate270CW	倍長整数	5	
TIFF Mirror Horizontal And Rotate90CW	倍長整数	7	
TIFF Mirror Vertical	倍長整数	4	
TIFF MM	倍長整数	4	
TIFF Next	倍長整数	32766	
TIFF Nikon NEF	倍長整数	34713	
TIFF None	倍長整数	1	
TIFF Pack Bits	倍長整数	32773	
TIFF Pentax PEF	倍長整数	65535	
TIFF Pixar Film	倍長整数	32908	
TIFF Pixar Log	倍長整数	32909	
TIFF Pixar Log L	倍長整数	32844	
TIFF Pixar Log Luv	倍長整数	32845	
TIFF RGB	倍長整数	2	
TIFF RGBPalette	倍長整数	3	
TIFF Rotate180	倍長整数	3	
TIFF Rotate270CW	倍長整数	8	
TIFF Rotate90CW	倍長整数	6	
TIFF SGILog	倍長整数	34676	
TIFF SGILog24	倍長整数	34677	
TIFF Sony ARW	倍長整数	32767	
TIFF T4Group3Fax	倍長整数	3	
TIFF T6Group4Fax	倍長整数	4	
TIFF Thunderscan	倍長整数	32809	
TIFF Transparency Mask	倍長整数	4	
TIFF UM	倍長整数	5	
TIFF Uncompressed	倍長整数	1	
TIFF White Is Zero	倍長整数	0	
TIFF YCb Cr	倍長整数	6	

Picture Transformation

定数	型	値	コメント
Crop	倍長整数	100	
Fade to grey scale	倍長整数	101	
Flip horizontally	倍長整数	3	
Flip vertically	倍長整数	4	
Horizontal concatenation	倍長整数	1	
Reset	倍長整数	0	
Scale	倍長整数	1	
Superimposition	倍長整数	3	
Translate	倍長整数	2	
Vertical concatenation	倍長整数	2	

Platform Interface

定数	型	値	コメント
Automatic Platform	倍長整数	-1	
Mac OS 7	倍長整数	0	
Mac OS 9	倍長整数	3	
Mac Theme	倍長整数	4	
Windows 3.11, NT 3.51	倍長整数	1	
Windows 9x	倍長整数	2	

Platform Properties

定数	型	値	コメント
_O_INTEL 386	倍長整数	386	
_O_INTEL 486	倍長整数	486	
_O_Macintosh 68K	倍長整数	1	
_O_PowerPC 601	倍長整数	601	
_O_PowerPC 603	倍長整数	603	
_O_PowerPC 604	倍長整数	604	
_O_PowerPC G3	倍長整数	510	
Intel Compatible	倍長整数	586	
Mac OS	倍長整数	2	
Power PC	倍長整数	406	
Windows	倍長整数	3	

Print Options

定数	型	値	コメント
Color option	倍長整数	8	
Destination option	倍長整数	9	
Double sided option	倍長整数	11	
Hide printing progress option	倍長整数	14	
Mac spool file format option	倍長整数	13	
Number of copies option	倍長整数	4	
Orientation option	倍長整数	2	
Paper option	倍長整数	1	
Paper source option	倍長整数	5	
PDFCreator Printer name	文字列	PDFCreator	
Scale option	倍長整数	3	
Spooler document name option	倍長整数	12	

Process State

定数	型	値	コメント
Aborted	倍長整数	-1	
Delayed	倍長整数	1	
Does not exist	倍長整数	-100	
Executing	倍長整数	0	
Hidden modal dialog	倍長整数	6	
Paused	倍長整数	5	
Waiting for input output	倍長整数	3	
Waiting for internal flag	倍長整数	4	
Waiting for user event	倍長整数	2	

Process Type

定数	型	値	コメント
_o_Web Process with Context	倍長整数	-11	
Apple Event Manager	倍長整数	-7	
Backup Process	倍長整数	-19	
Cache Manager	倍長整数	-4	
Client Manager Process	倍長整数	-31	
Created from execution dialog	倍長整数	3	
Created from Menu Command	倍長整数	2	
Design Process	倍長整数	-2	
Event Manager	倍長整数	-8	
Execute on Client Process	倍長整数	-14	
Execute on Server Process	倍長整数	1	
External Task	倍長整数	-9	
Indexing Process	倍長整数	-5	
Internal 4D Server Process	倍長整数	-18	
Internal Timer Process	倍長整数	-25	
Log File Process	倍長整数	-20	
Main Process	倍長整数	-1	
Method editor macro Process	倍長整数	-17	
Monitor Process	倍長整数	-26	
MSC Process	倍長整数	-22	
None	倍長整数	0	
On Exit Process	倍長整数	-16	
Other 4D Process	倍長整数	-10	
Other User Process	倍長整数	4	
Process Server Interface	倍長整数	-15	
Restore Process	倍長整数	-21	
Serial Port Manager	倍長整数	-6	
SQL Method Execution Process	倍長整数	-24	
Web Process on 4D Remote	倍長整数	-12	
Web Process with no Context	倍長整数	-3	
Web server Process	倍長整数	-13	

QR Area Properties

定数	型	値	コメント
qr view color toolbar	倍長整数	5	カラーツールバーのステータス取得 (表示=1, 非表示=0)
qr view column toolbar	倍長整数	6	カラムツールバーのステータス取得 (表示=1, 非表示=0)
qr view contextual menus	倍長整数	7	コンテキストメニューのステータス取得 (表示=1, 非表示=0)
qr view menubar	倍長整数	1	メニューバーのステータス取得 (表示=1, 非表示=0)
qr view operators toolbar	倍長整数	4	関数ツールバーのステータス取得 (表示=1, 非表示=0)
qr view standard toolbar	倍長整数	2	標準ツールバーのステータス取得 (表示=1, 非表示=0)
qr view style toolbar	倍長整数	3	スタイルツールバーのステータス取得 (表示=1, 非表示=0)

QR Borders

定数	型	値	コメント
qr bottom border	倍長整数	8	下罫線
qr inside horizontal border	倍長整数	32	内側縦罫線
qr inside vertical border	倍長整数	16	内側横罫線
qr left border	倍長整数	1	左罫線
qr right border	倍長整数	4	右罫線
qr top border	倍長整数	2	上罫線

QR Commands

定数	型	値	コメント
qr cmd 4D View destination	倍長整数	2503	
qr cmd add column	倍長整数	2608	
qr cmd alt back color palette	倍長整数	1004	
qr cmd automatic width	倍長整数	2605	
qr cmd average	倍長整数	507	
qr cmd back color palette	倍長整数	1003	
qr cmd back colors toolbar	倍長整数	2052	
qr cmd bold	倍長整数	500	
qr cmd borders	倍長整数	2609	
qr cmd center justified	倍長整数	504	
qr cmd columns toolbar	倍長整数	2054	
qr cmd count	倍長整数	510	
qr cmd default justified	倍長整数	512	
qr cmd delete column	倍長整数	2601	
qr cmd disk file destination	倍長整数	2501	
qr cmd edit column	倍長整数	2603	
qr cmd font color palette	倍長整数	1002	
qr cmd font dropdown	倍長整数	1000	
qr cmd format	倍長整数	2606	
qr cmd generate	倍長整数	2008	
qr cmd graph destination	倍長整数	2502	
qr cmd header and footer	倍長整数	2005	
qr cmd hide column	倍長整数	2602	
qr cmd hide line	倍長整数	2607	
qr cmd HTML file destination	倍長整数	2504	
qr cmd insert column	倍長整数	2600	
qr cmd italic	倍長整数	501	
qr cmd left justified	倍長整数	503	
qr cmd max	倍長整数	509	
qr cmd min	倍長整数	508	
qr cmd move left	倍長整数	3002	
qr cmd move right	倍長整数	3003	
qr cmd new	倍長整数	2000	
qr cmd open	倍長整数	2001	
qr cmd operators toolbar	倍長整数	2051	
qr cmd page setup	倍長整数	2006	
qr cmd plain	倍長整数	511	
qr cmd presentation	倍長整数	2611	
qr cmd print preview	倍長整数	2007	
qr cmd printer destination	倍長整数	2500	
qr cmd repeated values	倍長整数	2604	
qr cmd revert to save	倍長整数	2004	
qr cmd right justified	倍長整数	505	
qr cmd save	倍長整数	2002	
qr cmd save as	倍長整数	2003	
qr cmd standard deviation	倍長整数	513	
qr cmd standard toolbar	倍長整数	2053	
qr cmd style toolbar	倍長整数	2050	
qr cmd sum	倍長整数	506	
qr cmd totals spacing	倍長整数	2610	
qr cmd underline	倍長整数	502	

QR Document Properties

定数	型	値	コメント
qr printing dialog	倍長整数	1	プリントダイアログボックスの表示 • 値が1の場合、印刷の前に印刷ダイアログが表示されます。(デフォルト) • 値が0の場合、印刷の前に印刷ダイアログが表示されません。
qr unit	倍長整数	2	ドキュメントの単位 • 値が0の場合、ドキュメントの単位はポイントです。 • 値が1の場合、ドキュメントの単位はセンチです。 • 値が2の場合、ドキュメントの単位はインチです。

QR Operators

定数	型	値	コメント
qr average	倍長整数	2	
qr count	倍長整数	16	
qr max	倍長整数	8	
qr min	倍長整数	4	
qr standard deviation	倍長整数	32	
qr sum	倍長整数	1	

QR Output Destination

定数	型	値	コメント
_O_qr 4D Chart area	倍長整数	4	*** 廃止予定の定数 ***
qr 4D View area	倍長整数	3	N.A.
qr HTML file	倍長整数	5	ファイルへのパス名
qr printer	倍長整数	1	N.A.
qr text file	倍長整数	2	ファイルへのパス名

QR Report Types

定数	型	値	コメント
qr cross report	倍長整数	2	
qr list report	倍長整数	1	

QR Rows for Properties

定数	型	値	コメント
qr detail	倍長整数	-2	レポートの詳細エリア
qr footer	倍長整数	-5	ページフッタ
qr grand total	倍長整数	-3	総計エリア
qr header	倍長整数	-4	ページヘッダ
qr title	倍長整数	-1	レポートタイトル

QR Text Properties

定数	型	値	コメント
qr alternate background color	倍長整数	9	代替背景色
qr background color	倍長整数	8	背景色番号
qr bold	倍長整数	3	太字スタイル属性 (0 または 1)
qr font	倍長整数	1	Font numberが返すのと同様のフォント番号
qr font size	倍長整数	2	ポイント単位のフォントサイズ (9 ~ 255)
qr italic	倍長整数	4	イタリックスタイル属性 (0 または1)
qr justification	倍長整数	7	テキスト整列属性 (0 = デフォルト, 1 = 左, 2 = 中央, 3 = 右)
qr text color	倍長整数	6	フォントカラー属性 (カラー番号)
qr underline	倍長整数	5	下線スタイル属性 (0 または1)

Queries

定数	型	値	コメント
Description in Text Format	倍長整数	0	
Description in XML Format	倍長整数	1	
Into current selection	倍長整数	0	
Into named selection	倍長整数	2	
Into set	倍長整数	1	
Into variable	倍長整数	3	

Relations

定数	型	値	コメント
Automatic	倍長整数	3	カレントプロセスに対し、リレートを自動に設定する。
Do not modify	倍長整数	0	リレートの現在のステータスを変更しない。
Manual	倍長整数	2	カレントプロセスに対し、リレートをマニュアルに設定する。
No relation	倍長整数	0	
Structure configuration	倍長整数	1	アプリケーションのストラクチャウインドウで指定されたリレートの設定を使用する。

Resources Properties

定数	型	値	コメント
Changed resource bit	倍長整数	1	
Changed resource mask	倍長整数	2	
Locked resource bit	倍長整数	4	
Locked resource mask	倍長整数	16	
Preloaded resource bit	倍長整数	2	
Preloaded resource mask	倍長整数	4	
Protected resource bit	倍長整数	3	
Protected resource mask	倍長整数	8	
Purgeable resource bit	倍長整数	5	
Purgeable resource mask	倍長整数	32	
System heap resource bit	倍長整数	6	
System heap resource mask	倍長整数	64	

SCREEN DEPTH

定数	型	値	コメント
Black and white	倍長整数	0	
Four colors	倍長整数	2	
Is color	倍長整数	1	
Is gray scale	倍長整数	0	
Millions of colors 24 bit	倍長整数	24	
Millions of colors 32 bit	倍長整数	32	
Sixteen colors	倍長整数	4	
Thousands of colors	倍長整数	16	
Two fifty six colors	倍長整数	8	

SET RGB COLORS

定数	型	値	コメント
Background color	倍長整数	-2	
Background color none	倍長整数	-16	この定数は *backgroundColor* 引数と *altBackgrndColor* 引数にしか渡すことができません。
Dark shadow color	倍長整数	-3	
Disable highlight item color	倍長整数	-11	
Foreground color	倍長整数	-1	
Highlight menu background color	倍長整数	-9	
Highlight menu text color	倍長整数	-10	
Highlight text background color	倍長整数	-7	
Highlight text color	倍長整数	-8	
Light shadow color	倍長整数	-4	

Shortcut and Associated Keys

定数	型	値	コメント
Shortcut with Backspace	文字列	[backspace]	
Shortcut with Carriage Return	文字列	[return]	
Shortcut with Delete	文字列	[del]	
Shortcut with Down Arrow	文字列	[down arrow]	
Shortcut with End	文字列	[end]	
Shortcut with Enter	文字列	[enter]	
Shortcut with Escape	文字列	[esc]	
Shortcut with F1	文字列	[F1]	
Shortcut with F10	文字列	[F10]	
Shortcut with F11	文字列	[F11]	
Shortcut with F12	文字列	[F12]	
Shortcut with F13	文字列	[F13]	
Shortcut with F14	文字列	[F14]	
Shortcut with F15	文字列	[F15]	
Shortcut with F2	文字列	[F2]	
Shortcut with F3	文字列	[F3]	
Shortcut with F4	文字列	[F4]	
Shortcut with F5	文字列	[F5]	
Shortcut with F6	文字列	[F6]	
Shortcut with F7	文字列	[F7]	
Shortcut with F8	文字列	[F8]	
Shortcut with F9	文字列	[F9]	
Shortcut with Help	文字列	[help]	
Shortcut with Home	文字列	[home]	
Shortcut with Left Arrow	文字列	[left arrow]	
Shortcut with Page Down	文字列	[page down]	
Shortcut with Page Up	文字列	[page up]	
Shortcut with Right Arrow	文字列	[right arrow]	
Shortcut with Tabulation	文字列	[tab]	
Shortcut with Up Arrow	文字列	[up arrow]	

SQL

定数	型	値	コメント
SQL All Records	倍長整数	-1	
SQL Asynchronous	倍長整数	1	0 = 同期接続 (デフォルト値) 1 (または非0値) = 非同期接続
SQL Charset	倍長整数	100	(SQLパススルー経由で) 外部ソースに送られるリクエストで使用されるテキストエンコーディング。変更はカレントプロセスのカレント接続に対して実行されます。 値: MIBEnum識別子 (Note2を参照)、または値 -2 (Note3を参照) デフォルト: 106 (UTF-8)
SQL Connection Timeout	倍長整数	5	SQL LOGINコマンド実行時にレスポンスを待ち受ける最大タイムアウト。この値が有効になるためには、接続を開く前に設定しなければなりません。 取りうる値: 秒数 デフォルト: タイムアウトしない
SQL Max Data Length	倍長整数	3	返されるデータの最大長
SQL Max Rows	倍長整数	2	結果グループの最大行数 (プレビューで使用)
SQL On error abort	倍長整数	1	エラーイベント時、4Dは即座にスクリプト実行を停止します。
SQL On error confirm	倍長整数	2	エラーイベント時、4Dはエラーを説明するダイアログを表示し、スクリプト実行を停止するか続行するかをユーザーが選択できます。
SQL On error continue	倍長整数	3	エラーイベント時、4Dはそれを無視し、スクリプトの実行を続行します。
SQL Param In	倍長整数	1	
SQL Param In Out	倍長整数	2	SQLストアードプロシージャーのコンテキストでのみ利用可能です (ストアードプロシージャー内で定義されるin-outパラメーター)
SQL Param Out	倍長整数	4	SQLストアードプロシージャーのコンテキストでのみ利用可能です (ストアードプロシージャー内で定義されるoutパラメーター)
SQL Query Timeout	倍長整数	4	SQL EXECUTEコマンドの実行時に応答を待機する最大タイムアウト 値: 秒数 デフォルト:タイムアウトしない
SQL Use Access Rights	文字列	SQL_Use_Access_Rights	スクリプトのSQLコマンドの実行中に適用されるアクセス権を制限するために使用します。この属性を使用する場合、attribValueには1または0を渡します: • attribValue = 1: 4Dはカレントユーザーのアクセス権を使用します。 • attribValue = 0 (または属性を指定しない場合): 4Dはアクセスを制限しません。デザイナー権限が使用されます。
SQL_INTERNAL	文字列	;DB4D_SQL_LOCAL;	
System Data Source	倍長整数	2	
User Data Source	倍長整数	1	

Standard System Signatures

The Standard System Signatures are 4-character strings designated standard file types, resource types, standard data types stored into the Clipboard and so on.

定数	型	値	コメント
Picture Document	文字列	PICT	
Text Document	文字列	TEXT	
Windows MIDI Document	文字列	MID	
Windows Sound Document	文字列	WAV	
Windows Video Document	文字列	AVI	

System Documents

定数	型	値	コメント
Absolute path	倍長整数	2	documents配列は絶対パス名を格納
Alias selection	倍長整数	8	ドキュメントとしてショートカット(Windows) やエイリアス(Mac OS) を選択できます。この定数が使用されずにエイリアスやショートカットが選択されると、コマンドはターゲット要素のアクセスパスを返します。定数を渡すと、コマンドはエイリアスまたはショートカット自体のパスを返します。
Document unchanged	倍長整数	0	何も処理をしません。
Document with CR	倍長整数	3	改行は Mac OSフォーマット(CR、キャリッジリターン)へと変換されます。
Document with CRLF	倍長整数	2	改行はWindowsフォーマット(CRLF、キャリッジリターン＋ラインフィード)へと変換されます。
Document with LF	倍長整数	4	改行はUnixフォーマット(LF、ラインフィード)へと変換されます。
Document with native format	倍長整数	1	改行はOSのネイティブフォーマットに変換されます。Mac OS環境ではCR(キャリッジリターン)に、Windows環境ではCRLF(キャリッジリターン＋ラインフィード)に変換されます。(デフォルトの動作)
File name entry	倍長整数	32	ユーザーに"別名で保存 (新規保存)"ダイアログへのファイル名の入力を許可する。4Dはファイルを作成しません。ファイルの作成は開発者が行わなくてはなりません。Documentシステム変数が更新されます。このコンテキストではdirectory引数にディレクトリではなくファイルへのパスが格納されているかもしれません。そのファイル名は別名で保存テキストフィールドのデフォルトファイル名として使用されます。親ディレクトリはデフォルトパスとして使用されます。
Folder separator	文字列	Windows="\" Mac OS=":"	この定数は、実行中のOSに応じて異なる値を返します。この定数を使用すればプラットフォームの違いを気にせずにパス名を構築できます。 **注**: この定数はUnicodeモードの4Dアプリケーションでのみ使用できます。互換モードでは利用できません。
Get Pathname	倍長整数	3	
Ignore invisible	倍長整数	8	不可視ドキュメントをリストに含めない
Is a document	倍長整数	1	
Is a folder	倍長整数	0	
Multiple files	倍長整数	1	キーの組み合わせ**Shift+click** (隣接するファイルの選択) と**Ctrl+click** (Windows) 、または**Command+click** (Mac OS) を使用すると、複数のファイルを同時に選択できます。この場合引数selectedを渡し、選択されたファイルをすべてリストにして受け取ります。この定数が使用されないと、コマンドは複数ファイルの選択を許可しません。
Package open	倍長整数	2	(Mac OSのみ): パッケージをフォルダとして開きその内容を閲覧できます。この定数が使用されないと、コマンドはパッケージの開封を許可しません。
Package selection	倍長整数	4	(Mac OSのみ): パッケージの選択を許可します。この定数が使用されないと、コマンドはソフトウェアパッケージの選択を許可しません。
Posix path	倍長整数	4	documents配列はPOSIXフォーマットパス名を格納
Read and Write	倍長整数	0	デフォルトモード
Read Mode	倍長整数	2	
Recursive parsing	倍長整数	1	documents配列は指定したフォルダーに含まれるすべてのファイルとサブフォルダーを格納
Use sheet window	倍長整数	16	(Mac OSのみ): 選択ダイアログボックスをシートウィンドウで表示します(Windowsでは、このオプションは無効です)。シートウィンドウはMac OS Xインタフェースに特有なもので、グラフィックアニメーションを伴います (詳細については、ウィンドウタイプを参照してください) 。この定数が使用されないと、コマンドは標準なダイアログボックスを表示します。
Write Mode	倍長整数	1	

System Folder

定数	型	値	コメント
_O_Mac Control Panels	倍長整数	11	
_O_Mac Extensions	倍長整数	10	
_O_Mac Shutdown Items_All	倍長整数	6	
_O_Mac Shutdown Items_User	倍長整数	7	
Applications or Program Files	倍長整数	16	
Desktop	倍長整数	15	
Documents folder	倍長整数	17	OSユーザーのDocumentsフォルダー
Favorites Win	倍長整数	14	
Fonts	倍長整数	1	
Start Menu Win_All	倍長整数	8	
Start Menu Win_User	倍長整数	9	
Startup Win_All	倍長整数	4	
Startup Win_User	倍長整数	5	
System	倍長整数	0	
System Win	倍長整数	12	
System32 Win	倍長整数	13	
User Preferences_All	倍長整数	2	
User Preferences_User	倍長整数	3	

System Format

定数	型	値	コメント
Currency symbol	倍長整数	2	通貨記号 (例: “¥”)
Date separator	倍長整数	13	日付フォーマットの区切り文字 (例: “/”)
Decimal separator	倍長整数	0	小数区切り文字 (例: “.”)
Short date day position	倍長整数	15	短日付フォーマットでの日の位置: “1” = 左, “2” = 中央, “3” = 右
Short date month position	倍長整数	16	短日付フォーマットでの月の位置: “1” = 左, “2” = 中央, “3” = 右
Short date year position	倍長整数	17	短日付フォーマットでの年の位置: “1” = 左, “2” = 中央, “3” = 右
System date long pattern	倍長整数	8	“dddd MMMM yyyy”形式に対応する長日付表示フォーマット
System date medium pattern	倍長整数	7	“dddd MMMM yyyy”形式に対応する日付表示フォーマット
System date short pattern	倍長整数	6	“dddd d MMMM yyyy”形式に対応する日付表示フォーマット
System time AM label	倍長整数	18	12時間フォーマット時に午前を示すラベル
System time long pattern	倍長整数	5	“HH:MM:SS”形式に対応する長時間表示フォーマット
System time medium pattern	倍長整数	4	“HH:MM:SS”形式に対応する時間表示フォーマット
System time PM label	倍長整数	19	12時間フォーマット時に午後を示すラベル
System time short pattern	倍長整数	3	“HH:MM:SS”形式に対応する時間表示フォーマット
Thousand separator	倍長整数	1	千の位区切り文字 (例: “,”)
Time separator	倍長整数	14	時間フォーマットの区切り文字 (例: “:”)

TCP Port Numbers

定数	型	値	コメント
TCP Authentication	倍長整数	113	
TCP DNS	倍長整数	53	
TCP Finger	倍長整数	79	
TCP FTP Control	倍長整数	21	
TCP FTP Data	倍長整数	20	
TCP Gopher	倍長整数	70	
TCP HTTP WWW	倍長整数	80	
TCP IMAP3	倍長整数	220	
TCP Kerberos	倍長整数	88	
TCP KLogin	倍長整数	543	
TCP Nickname	倍長整数	43	
TCP NNTP	倍長整数	119	
TCP NTalk	倍長整数	518	
TCP NTP	倍長整数	123	
TCP PMCP	倍長整数	1643	
TCP PMD	倍長整数	1642	
TCP POP3	倍長整数	110	
TCP Printer	倍長整数	515	
TCP RADACCT	倍長整数	1646	
TCP RADIUS	倍長整数	1645	
TCP Remote Cmd	倍長整数	514	
TCP Remote Exec	倍長整数	512	
TCP Remote Login	倍長整数	513	
TCP Router	倍長整数	520	
TCP SMTP	倍長整数	25	
TCP SNMP	倍長整数	161	
TCP SNMPTRAP	倍長整数	162	
TCP SUN RPC	倍長整数	111	
TCP Talk	倍長整数	517	
TCP Telnet	倍長整数	23	
TCP TFTP	倍長整数	69	
TCP UUCP	倍長整数	540	
TCP UUCP RLOGIN	倍長整数	541	

Time Display Formats

定数	型	値	コメント
Blank if null time	倍長整数	100	0の代わりに""
HH MM	倍長整数	2	01:02
HH MM AM PM	倍長整数	5	1:02 AM
HH MM SS	倍長整数	1	01:02:03
Hour Min	倍長整数	4	1時2分
Hour Min Sec	倍長整数	3	1時2分3秒
ISO Time	倍長整数	8	0000-00-00T01:02:03
Min Sec	倍長整数	7	62分3秒
MM SS	倍長整数	6	62:03
System time long	倍長整数	11	1:02:03 AM HNEC (Macのみ)
System time long abbreviated	倍長整数	10	1:02:03 AM (Macのみ)
System time short	倍長整数	9	01:02:03

Trigger Events

定数	型	値	コメント
_O_On Loading Record Event	倍長整数	4	
On Deleting Record Event	倍長整数	3	
On Saving Existing Record Event	倍長整数	2	
On Saving New Record Event	倍長整数	1	

Value for Associated Standard Action

定数	型	値	コメント
Accept Action	倍長整数	2	
Add subrecord Action	倍長整数	14	
Cancel Action	倍長整数	1	
Clear Action	倍長整数	21	
Copy Action	倍長整数	19	
Cut Action	倍長整数	18	
Database Settings Action	倍長整数	32	
Delete record Action	倍長整数	7	
Delete subrecord Action	倍長整数	13	
Edit subrecord Action	倍長整数	12	
First page Action	倍長整数	10	
First record Action	倍長整数	5	
Last page Action	倍長整数	11	
Last record Action	倍長整数	6	
MSC Action	倍長整数	36	
Next page Action	倍長整数	8	
Next record Action	倍長整数	3	
No Action	倍長整数	0	
Paste Action	倍長整数	20	
Previous page action	倍長整数	9	
Previous record Action	倍長整数	4	
Quit Action	倍長整数	27	
Redo Action	倍長整数	31	
Return to Design mode	倍長整数	35	
Select all Action	倍長整数	22	
Show Clipboard Action	倍長整数	23	
Test Application Action	倍長整数	26	
Undo Action	倍長整数	17	

Web Area

定数	型	値	コメント
WA enable contextual menu	倍長整数	4	Webエリア内で標準のコンテキストメニューの表示を許可する
WA enable Java applets	倍長整数	1	Webエリア内でJava appletの実行を許可する
WA enable JavaScript	倍長整数	2	Webエリア内でJavaScriptコードの実行を許可する
WA enable plugins	倍長整数	3	Webエリア内でプラグインのインストールを許可する
WA next URLs	倍長整数	1	
WA previous URLs	倍長整数	0	

Web Server

定数	型	値	コメント
Web Character set	倍長整数	17	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: データベースに接続するブラウザーとの通信に (ローカルモード4Dと4D Serverの) Webサーバーが使用する文字セット。デフォルト値はOSの言語に依存します。この引数はデータベース設定でも設定できます。 **値**: 取りうる値は、文字セットに関連するデータベースの動作モードによります。 • Unicode モード: アプリケーションがUnicodeモードで動作している場合、この引数に渡す値は文字セット識別子です。(*MIBEnum*倍長整数または文字列。以下のアドレスを参照: http://www.iana.org/assignments/character-sets)。以下は4D Webサーバがサポートする文字セットに対応する識別子のリストです: 4=ISO-8859-1 12=ISO-8859-9 13=ISO-8859-10 17=Shift_JIS 2024=Windows-31J 2026=Big5 38=euc-kr 106=UTF-8 2250=Windows-1250 2251=Windows-1251 2253=Windows-1253 2255=Windows-1255 2256=Windows-1256 **注**: IANAに定義されていない特別な文字セット (1258=x-mac-japanese) を使用することができます。これはWindows上ではコードページ10001に、Mac上ではkTextEncodingMacJapaneseにマップされています。 • *ASCII 互換モード*: 0: Western European 1: Japanese 2: Chinese 3: Korean 4: User-defined 5: Reserved 6: Central European 7: Cyrillic 8: Arabic 9: Greek 10: Hebrew 11: Turkish 12: Baltic
Web HTTP Compression Level	倍長整数	50	**スコープ**: *ローカルWebサーバー* **2セッション間で設定を保持**: No **説明**: 4D HTTPサーバーで使用されるすべての圧縮されたHTTP通信 (クライアントのリクエスト、サーバーのレスポンス、WebおよびWebサービス) の圧縮レベル。このセレクターを使用すれば圧縮率を犠牲にして実行速度を速めるか、速度を優先して圧縮率を高めるかを選択できます。値の選択は交換するデータのサイズやタイプに基づきます。value引数に1から9までの値を渡します。1は速度優先、9は圧縮率優先です。また-1を渡して圧縮速度と圧縮率の妥協点を指定できます。デフォルトの圧縮レベルは1 (速度優先) です。 **とりうる値**: 1 から 9 (1 = 速度優先, 9 = 圧縮優先) または -1 = 最適
Web HTTP Compression Threshold	倍長整数	51	**スコープ**: ローカルHTTPサーバー **2セッション間で設定を保持**: No **説明**: 最適化モードの内部的な4D Webサービス通信フレームワークにおいて、圧縮を行わないリクエストサイズの敷居値を設定できます。この設定は、小さなデータ交換時に圧縮を行うことによる、マシンの時間の浪費を避けるために有効です。 **とりうる値**: 任意の倍長整数値。valueにはバイト単位でサイズを渡します。デフォルトで、圧縮の敷居値は1024バイトに設定されています。
Web HTTPS Port ID	倍長整数	39	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: ローカルモード4Dおよび4D ServerのWebサーバがSSLによるセキュアな接続(HTTPSプロトコル)で使用するTCPポート番号。HTTPS ポート番号はデータベース設定の"Web/設定"ページで指定できます。デフォルト値は443 (標準ポート番号) です。value引数にTCP Port Numbersテーマの定数を渡すこともできます。 **とりうる値**: 0~65535
Web Inactive process timeout	倍長整数	78	**スコープ**: ローカルWebサーバー **2セッション間で値を保持**: No. しかしHTTPサーバーを再起動しただけの場合は保持されます。 **説明**: セッション管理のために使用されるプロセスのタイムアウトを設定します。タイムアウト後、プロセスは終了されます。 **取りうる値**: 倍長整数 (分) **デフォルト値**: 480分 (= 8時間) (0を渡すとデフォルト値に設定されます)
Web Inactive session timeout	倍長整数	72	**スコープ**: ローカルWebサーバー **2セッション間で値を保持**: No. しかしHTTPサーバーを再起動しただけの場合は保持されます。 **説明**: セッション管理のために使用されるcookieのタイムアウトを設定します。 **取りうる値**: 倍長整数 (分) **デフォルト値**: 480分 (= 8時間) (0を渡すとデフォルト値に設定されます)

Web IP Address to listen	倍長整数	16	**スコープ**: 4D ローカル, 4D Server
			2セッション間で設定を保持: Yes
			説明: ローカルモード4Dおよび4D ServerのWebサーバがHTTPリクエストを受信するIPアドレス。デフォルトで、特定のアドレスは定義されていません(value=0)。この引数はデータベース設定で設定できます。
			Web IP Address to listenセレクターは、コンパイルして4D Volume Desktopを組み込んだ4D Webサーバで役立ちます (この場合デザインモードへのアクセス手段がありません)。
			value引数には16進数のIPアドレスを渡します。つまり、"a.b.c.d"のようなIPアドレスを指定するには、以下のようなコードを作成します:
			`C_LONGINT($addr)`
			`$addr:=($a<<24)|($b<<16)\|($c<<8)|$d`
			`SET DATABASE PARAMETER(Web IP Address to listen;$addr)`

Web Keep session	倍長整数	70	**スコープ**: ローカルWebサーバー **2セッション間で設定を保持**: No. しかしHTTPサーバー再起動後も設定は有効。 **説明**: 4Dによる自動セッション管理モード (Webセッション管理) の有効/無効を設定する。 **取りうる値**: 1 (有効) / 0 (無効) **デフォルト値**: v13で作成されたデータベースでは1、変換されたデータベースでは0。このモードはリモートモードで一時的なコンテキストの再利用メカニズムも有効にする点に留意してください。このメカニズムに関する詳細はWebサーバー設定を参照してください。
Web Log Recording	倍長整数	29	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: ローカルモード4Dまたは4D ServerのWebサーバーが受け取るWebリクエストの記録を開始または停止します。デフォルト値は0(リクエストを記録しない)です。 Web リクエストのログは"logweb.txt"という名前のテキストファイルに保存されます。このファイルは自動でストラクチャファイルと同階層のLogsフォ ルダー内に作成されます。このファイルのフォーマットは、渡した値により決定されます。Webログファイルフォーマットに関する詳細はWebサイトに関する情報を参照してください。 このファイルはデータベース設定の"Web/ログ"ページからも有効にできます。 **とりうる値**: 0 = 記録しない (デフォルト), 1 = CLFフォーマットで記録, 2 = DLFフォーマットで記録, 3 = ELFフォーマットで記録, 4 = WLFフォーマットで記録 **警告**: フォーマット3および4はカスタムフォーマットであり、記録する内容を事前にデータベース設定で定義しなければなりません。事前定義せずにこれらのフォーマットを使用した場合、ログファイルは作成されません。
Web Max Concurrent Processes	倍長整数	18	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: ローカルモード4Dならびに4D ServerのWebサーバーでサポートされる、任意のタイプの同時Webプロセス上限数を厳密に設定。この上限数(マイナス1)に達した場合、4Dはそれ以上プロセスを作成しなくなり、HTTPステータス503(Service Unavailable)をすべての新しいリクエストに返します。 このパラメーターにより、同時に行われる非常に膨大な数のリクエストやコンテキスト作成に関する過大な要求の結果として、4D Webサーバが飽和状態になることを防ぐことができます。このパラメーターはデータベース設定でも設定できます。 理論上、Webプロセスの最大数は次の計算式の結果になります: 使用可能メモリ/Webプロセスのスタックサイズ。別の解決策は、ランタイムエクスプローラに表示されるWebプロセス情報を監視する方法です。つまり現在のWebプロセス数およびWebサーバの開始以降に達した最大数を監視します。 **値**: 10から32 000までの任意の数。デフォルト値は100。
Web Max sessions	倍長整数	71	**スコープ**: ローカルWebサーバー **2セッション間で設定を保持**: No. しかしHTTPサーバー再起動後も設定は有効。 **説明**: 4Dの自動セッション管理下のセッション上限数。設定した上限に達すると、もっとも古いセッションが閉じられます。 **取りうる値**: 倍長整数値 **デフォルト値**: 100 (0を渡すとデフォルト値が設定されます)
Web Maximum requests size	倍長整数	27	**スコープ**: 4D ローカル, 4D Server **2セッション間で設定を保持**: Yes **説明**: Webサーバに処理を許可する受信HTTPリクエスト (POST) の最大サイズ (バイト単位)。デフォルト値は2,000,000 (2MBより少し少ない値) です。最大値(2,147,483,648)を渡すと、実際上は制限がなくなります。 この制限は、受信するリクエストが大きすぎるためにWebサービスが飽和してしまうことを回避するために使用します。リクエストがこの制限に達すると、4D Webサービスはリクエストを拒否します。 **とりうる値**: 500,000～2,147,483,648.
Web Session cookie name	倍長整数	73	スコープ: ローカルWebサーバー 2セッション間で設定を保持: No. しかしHTTPサーバー再起動後も設定は保持される。 説明: 4Dの自動セッション管理機能で使用されるcookieのname属性値。 取りうる値: テキスト デフォルト値: "4DSID" (空文字を渡すとデフォルト値が設定されます。)

Web Services (Client)

定数	型	値	コメント
Web Service Compression	倍長整数	1	
Web Service Detailed Message	倍長整数	1	エラーの説明メッセージ。メインのエラータイプに応じてメッセージは異なります。 - 9910 (Soap fault): SOAPエラーの原因が返されます (例: "リモートメソッドが存在しません")。 - 9911 (Parser fault): XMLドキュメント中のエラーの個所が返されます。 - 9912 (HTTP fault): - HTTPエラーが [300-400] の場合 (問題がリクエストされたドキュメントの場所に関連する場合)、リクエストURLの新しい場所が返されます。 - 他のHTTPエラーコードの場合、<body>が返されます。 - 9913 (Network fault): ネットワークエラーの原因が返されます (例: "ServerAddress: DNS lookup failure") - 9914 (Internal fault): 内部エラーの原因が返されます。
Web Service display auth dialog	倍長整数	4	value = 0 (ダイアログボックスを表示しない) または 1 (ダイアログボックスを表示する) このオプションはWEB SERVICE CALLコマンド実行時の認証ダイアログボックスの表示を管理します。デフォルトでダイアログボックスは表示されません。通常WEB SERVICE AUTHENTICATEコマンドを使用して認証を行わなければなりません。しかし認証ダイアログボックスを表示してユーザに認証情報を入力させたい場合、このオプションを使用します。value に1を渡すとダイアログを表示、0を渡すと表示しません。ダイアログボックスはWebサービスが認証を要求する場合のみ表示されます。
Web Service Dynamic	倍長整数	0	
Web Service Error Code	倍長整数	0	(4Dが定義した) 主たるエラーコード。このコードはErrorシステム変数にも返されます。 返される可能性のあるコードは以下のとおりです: 9910: Soap fault (参照 Web Service Fault Actor) 9911: Parser fault 9912: HTTP fault (参照 Web Service HTTP Error code) 9913: Network fault 9914: Internal fault.
Web Service Fault Actor	倍長整数	3	エラーの原因 (SOAPプロトコルにより返される、メインエラーが9910の場合に使用)。 - バージョンが合わない - 引数解釈エラー (必須として定義された引数をサーバが解釈できない) - 送信側側のエラー - 受信側のエラー - 未知のエンコーディング
Web Service HTTP Compression	倍長整数	6	value = Web Service Compression このオプションは、SOAPリクエストによる4Dアプリケーション間のデータ交換を高速化する目的で、内部的な圧縮メカニズムを有効にするために使用します。WEB SERVICE SET OPTION(Web Service HTTP Compression; Web Service Compression) ステートメントをWebサービスの4Dクライアント上で実行すると、クライアントから送信される次のSOAPリクエストのデータは、4D SOAPサーバーに送信される前に、標準のHTTPメカニズム (リクエストの内容に応じて"gzip" または "deflate") を使用して圧縮されます。サーバーはリクエストを解凍・解析し、自動で同じメカニズムを使用して応答します。WEB SERVICE SET OPTIONコマンドの呼出し直後のリクエストのみに影響します。すなわち圧縮を使用したいリクエスト毎にこのコマンドを呼び出す必要があります。デフォルトで4DはWebサービスHTTPリクエストを圧縮しません。 **注:** • このメカニズムは11.3以前の4D SOAPサーバーへのリクエストでは利用できません。 • この機能を最適化するために、リクエストの圧縮を行う閾値や圧縮率を設定する追加のオプションがあります。これらのオプションを設定するにはSET DATABASE PARAMETERコマンドを使用します。
Web Service HTTP Error code	倍長整数	2	HTTPエラーコード (メインエラーが9912の場合に使用)。
Web Service HTTP Timeout	倍長整数	1	value = 秒単位で指定するクライアント側のタイムアウト クライアント側のタイムアウトは、サーバが返答しない場合のWebサービスクライアント側の待ち時間です。この時間経過後、クライアントはセッションを閉じ、リクエストは失われます。 このタイムアウトはデフォルトで180秒です。特定の理由 (ネットワークの状態、Webサービスの仕様等) でこの値を変更できます。
Web Service Manual	倍長整数	3	
Web Service Manual In	倍長整数	1	
Web Service Manual Out	倍長整数	2	
Web Service reset auth settings	倍長整数	5	value = 0 (情報を消去しない) または 1 (情報を消去する) このオプションを使用して、4Dにユーザの認証情報 (ユーザ名とパスワード、認証メソッド等) を記憶させ、それを再利用するかどうかを指定できます。デフォルトでこの情報はWEB SERVICE CALLコマンドを呼び出すたびに消去されます。value に0を渡すと情報は保持され、1を渡すと消去されます。0を渡した場合、情報はセッションの間保持されます。
Web Service SOAP Header	倍長整数	2	value = SOAPリクエストのヘッダとして挿入するXMLルート要素参照 このオプションを使用して、WEB SERVICE CALLコマンドで生成されるSOAPリクエストにヘッダを挿入できます。デフォルトでSOAPリクエストは特定のヘッダを持っていません。しかしWebサービスによっては、例えば識別情報を管理するために、ヘッダを要求することがあります。
Web Service SOAP Version	倍長整数	3	value = Web Service SOAP_1_1 または Web Service SOAP_1_2 このオプションで、リクエストで使用するSOAPプロトコルのバージョンを指定できます。Web Service SOAP_1_1 定数を value に渡すとバージョン1.1が、Web Service SOAP_1_2 を渡すとバージョン1.2が指定されます。
Web Service SOAP_1_1	倍長整数	0	
Web Service SOAP_1_2	倍長整数	1	

Web Services (Server)

定数	型	値	コメント
Is DOM reference	倍長整数	37	
Is XML	倍長整数	36	
SOAP Client Fault	倍長整数	1	
SOAP Input	倍長整数	1	
SOAP Method Name	倍長整数	1	実行されようとしているWebサービスメソッド名
SOAP Output	倍長整数	2	
SOAP Server Fault	倍長整数	2	
SOAP Service Name	倍長整数	2	メソッドが属しているWebサービスの名前

Window Kind

定数	型	値	コメント
External window	倍長整数	5	
Floating window	倍長整数	14	
Modal dialog	倍長整数	9	
Regular window	倍長整数	8	

XML

定数	型	値	コメント
Copy XML Data Source	倍長整数	1	4Dはピクチャとともに DOMツリーのコピーを保持します。ピクチャをデータベースのピクチャフィールドに保存して、いつでも再び表示したり、書きだしたりできます。
DOCTYPE Name	倍長整数	3	DOCTYPE マーカで定義されているルート要素の名前。
Document URI	倍長整数	6	DTDのURI
Encoding	倍長整数	4	使用されているエンコーディング (UTF-8, ISO...).
Get XML Data Source	倍長整数	0	4DはXMLデータソースの読み込みのみを行います。XMLデータソースはピクチャと一緒には保持されません。これはコマンドの実行速度を大幅に向上させますが、DOMツリーが保持されないため、ピクチャを格納したり書きだしたりすることはできません。
Own XML Data Source	倍長整数	2	4DはDOMツリーをピクチャとともに書き出します。ピクチャを格納したり書きだしたりでき、かつコマンドの実行も速いです。しかしelementRef XML参照を他の4Dコマンドで使用することはできなくなります。これはexportType 引数が省略された場合のデフォルトモードです。
PUBLIC ID	倍長整数	1	ドキュメントが従うDTDの公開識別子 (FPI, DOCTYPE xxx PUBLICタグが存在する場合)。
SYSTEM ID	倍長整数	2	システム識別子。
Version	倍長整数	5	受け入れられたXMLバージョン。
XML Base64	倍長整数	1	
XML Binary encoding	倍長整数	5	バイナリデータを変換する方法を指定します。 とりうる値は: • XML Base64 (デフォルト値): バイナリデータは単純にBase64に変換される • XML Data URI scheme: バイナリデータはBase64に変換され、"data:;base64"ヘッダが追加される。このフォーマットは主に、ブラウザが自動でピクチャをデコードできるようにするために使用されます。またSVGピクチャの挿入にも必要です。詳細はhttp://en.wikipedia.org/wiki/Data_URI_schemeを参照してください。
XML CDATA	倍長整数	7	
XML Comment	倍長整数	2	
XML Convert to PNG	倍長整数	1	
XML DATA	倍長整数	6	
XML Data URI scheme	倍長整数	2	
XML Date encoding	倍長整数	2	4D日付の変換方法を指定します。例えば日本のタイムゾーンで !2003/01/01! の例で、とりうる値は (時差によりUTCでは日付が異なる場合があります): • XML ISO (デフォルト値): タイムゾーンの指定なしでxs:datetimeフォーマットを使用します。結果は: "2003-01-01"。時間部がSQLなどを使用して4D値に含まれていても、それは失われます。 • XML Local: タイムゾーンを指定してxs:dateフォーマットを使用します。結果は: "2003-01-01 +09:00"。時間部がSQLなどを使用して4D値に含まれていても、それは失われます。 • XML Datetime local: タイムゾーンを指定してxs:dateTime (ISO 8601) フォーマットを使用します。時間部がSQLなどを使用して4D値に含まれている場合、このフォーマットでは時間部が保持されます。結果は: "<Date>2003-01-01T00:00:00 +09:00</Date>"。 • XML UTC: xs:dateフォーマットを使用します。結果は: "2003-01-01Z"。時間部がSQLなどを使用して4D値に含まれていても、それは失われます。 • XML Datetime UTC: xs:dateTime (ISO 8601)フォーマットを使用します。時間部がSQLなどを使用して4D値に含まれている場合、このフォーマットでは時間部が保持されます。結果は: "<Date>2003-01-01T00:00:00Z</Date>".
XML Datetime local	倍長整数	3	
XML Datetime local absolute	倍長整数	1	
XML Datetime UTC	倍長整数	5	
XML DOCTYPE	倍長整数	10	
XML Duration	倍長整数	2	
XML ELEMENT	倍長整数	11	
XML End Document	倍長整数	9	
XML End Element	倍長整数	5	
XML Entity	倍長整数	8	
XML Indentation	倍長整数	4	XMLドキュメントのインデントを指定します。 とりうる値: • XML With indentation (デフォルト値): ドキュメントはインデントされる • XML No indentation: ドキュメントはインデントされない。内容は一行中に置かれます。
XML ISO	倍長整数	1	
XML Local	倍長整数	2	
XML Native codec	倍長整数	2	
XML No indentation	倍長整数	2	
XML Picture encoding	倍長整数	6	(Base64にエンコードされる前に) ピクチャの変換の方法を指定します。 とりうる値: • XML Convert to PNG (デフォルト値): Base64にエンコードされる前に、ピクチャはPNGに変換されます。 • XML Native codec: Base64にエンコードされる前に、ピクチャは最初のネイティブなストレージCODECに変換されます。SVGピクチャをエンコードするためにこれらのオプションを使用しなければなりません (XML SET OPTIONSコマンドの例題参照)。
XML Processing Instruction	倍長整数	3	
XML Raw data	倍長整数	2	
XML Seconds	倍長整数	4	
XML Start Document	倍長整数	1	
XML Start Element	倍長整数	4	
XML String encoding	倍長整数	1	4D文字列を要素値に変換する方法を指定します。これはXMLでエスケープ文字の利用が必須である属性の変換には影響しません。 とりうる値: • XML With escaping (デフォルト値): 4D文字列をXML要素値に変換する際、文字の置き換えを行います。テキスト型のデータは自動で解析され、禁止されている文字 (<&>") はXML実体参照 (&<> ") に置き換えられます。 • XML Raw data: 4D文字列は生データとして送信されます。4Dはエンコードや解析を行いません。4Dの値は可能であればXMLフラグメントに変換され、ターゲット要素の子要素として挿入されます。値をXMLフラグメントとして扱えない場合、新しいCDATAノードに生データとして挿入されます。
XML Time encoding	倍長整数	3	4Dの時間を変換する方法を指定します。例: ?02/00/46? (日本時間)。エンコーディングは時刻を表すか時間を表すかにより異なります。 時刻の場合: • XML Datetime UTC: UTC (Universal Time Coordinated) で表現された時刻。UTCへの変換は自動です。結果: "<Duration>0000-00-00T17:00:46Z</Duration>". • XML Datetime local: 時刻は4Dエンジンが実行されているマシンの時差を使用して表現されます。結果: "<Duration>0000-00-00T02:00:46+09:00</Duration>". • XML Datetime local absolute (デフォルト値): 時刻は時差なしで表現されます。値は変更されません。結果: "<Duration>0000-00-00T02:00:46</Duration>". 時間の場合: • XML Seconds: 00:00:00からの経過秒数。時間をあらわすため、値は変更されません。結果: "<Duration>7246</Duration>". • XML Duration: XML Schema Part 2に準拠した時間表現。時間をあらわすため、値は変更されません。結果: "<Duration>PT02H00M46S</Duration>".
XML UTC	倍長整数	4	
XML With escaping	倍長整数	1	
XML With indentation	倍長整数	1	

関連付けられた標準アクションのテキスト値

定数	型	値	コメント
Object Accept action	文字列	2	
Object Add subrecord action	文字列	14	
Object Automatic splitter action	文字列	16	
Object Cancel action	文字列	1	
Object Clear action	文字列	21	
Object Copy action	文字列	19	
Object Cut action	文字列	18	
Object Database Settings action	文字列	32	
Object Delete record action	文字列	7	
Object Delete subrecord action	文字列	13	
Object Edit subrecord action	文字列	12	
Object First page action	文字列	10	
Object First record action	文字列	5	
Object Goto page action	文字列	15	
Object Last page action	文字列	11	
Object Last record action	文字列	6	
Object MSC action	文字列	36	
Object Next page action	文字列	8	
Object Next record action	文字列	3	
Object No standard action	文字列	0	
Object Open back URL action	文字列	37	
Object Open next URL action	文字列	38	
Object Paste action	文字列	20	
Object Previous page action	文字列	9	
Object Previous record action	文字列	4	
Object Quit action	文字列	27	
Object Redo action	文字列	31	
Object Refresh current URL action	文字列	39	
Object Return to Design mode action	文字列	35	
Object Select all action	文字列	22	
Object Show Clipboard action	文字列	23	
Object Stop loading URL action	文字列	40	
Object Test Application action	文字列	26	
Object Undo action	文字列	17	

エラーコード

- シンタックスエラー (1 -> 81)
- データベースエンジンエラー (-10600 -> 4004)
- SQLエンジンエラー (1001 -> 3018)
- ネットワークエラー (-10051 -> -10001)
- バックアップマネージャエラー (1401 -> 1421)
- OSファイルマネージャエラー (-124 -> -33)
- OSメモリマネージャエラー (-117 -> -108)
- OSプリントマネージャエラー (-8192 -> -1)
- OSリソースマネージャエラー (-196 -> -1)
- SANE NaNエラー (1 -> 255)
- OSサウンドマネージャエラー (-209 -> -203)
- OSシリアルポートマネージャエラー (-28)
- Mac OSシステムエラー (4 -> 28)

シンタックスエラー (1 -> 81)

以下の表にはデザインまたはアプリケーション環境でコードを実行中に発生するシンタックスエラーコードをリストします。いくつかのエラーはインタプリタモードでのみ、いくつかはコンパイルモードでのみ、そしていくつかは両方で発生します。これらのエラーはON ERR CALLでインストールされるエラー処理メソッドでとらえることができます。

コード	説明
1	"(" が必要です。
2	フィールドが必要です。
3	このコマンドはサブテーブルのフィールドに対してのみ実行できます。
4	リスト中の引数はすべて同じタイプでなければなりません。
5	このコマンドの対象となるテーブルがありません。
6	このコマンドはサブテーブル型のフィールドに対してのみ実行できます。
7	数値引数が必要です。
8	文字引数が必要です。
9	条件テストの結果が必要です。
10	このコマンドはこのフィールドタイプに使用できません。
11	このコマンドは2つの条件判断には使用できません。
12	このコマンドは2つの数値引数には使用できません。
13	このコマンドは2つの文字引数には使用できません。
14	このコマンドは2つの日付引数には使用できません。
15	この処理はこれら2つの引数に対応していません。
16	フィールドにリレーションがありません。
17	テーブルが必要です。
18	フィールドタイプが対応していません。
19	フィールドにインデックスが設定されていません。
20	"="が必要です。
21	メソッドが存在しません。
22	ソートやグラフに用いるフィールドは同じテーブルまたはサブテーブルに属していなければなりません。
23	"<" または ">" が必要です。
24	";" が必要です。
25	ソートするフィールドが多すぎます。
26	フィールドタイプがテキスト、ピクチャ、Blob、またはサブテーブルであってはなりません。
27	フィールドはテーブル名の後に書かなければなりません。
28	フィールドは数値型でなければなりません。
29	値は1 または 0でなければなりません。
30	変数が必要です。
31	この番号のメニューバーがありません。
32	日付が必要です。
33	実装されていないコマンドまたは関数です。
34	計算ファイルが開かれていません。
35	セットが異なるテーブルに属しています。
36	無効なテーブル名です。
37	":=" が必要です。
38	これは関数です。メソッドではありません。
39	セットが存在しません。
40	これはメソッドです。関数ではありません。
41	サブテーブルに属する変数またはフィールドが必要です。
42	このレコードをスタックにプッシュできません。
43	関数が見つかりません。
44	メソッドが見つかりません。
45	フィールドまたは変数が必要です。
46	数値または文字型の引数が必要です。
47	フィールドタイプは文字型でなければなりません。
48	シンタックスエラー。
49	この処理をここで使用することはできません。
50	これらの処理を一緒に使用することはできません。
51	実装されていないモジュールです。
52	配列が必要です。
53	インデックスが範囲外です。
54	引数のタイプが対応していません。
55	ブール引数が必要です。
56	フィールド、変数、またはテーブルが必要です。
57	演算子が必要です。
58	")" が必要です。
59	この種類の引数はここでは使用できません。
60	引数またはローカル変数はコンパイルされたデータベースでのEXECUTE文では使用できません。
61	コンパイルされたデータベースでは配列のタイプを変更できません。
62	このコマンドはサブテーブルには使用できません。
63	このフィールドはインデックスが付けられていません。
64	ピクチャフィールドまたは変数が必要です。
65	値には4文字が必要です。
66	値は4文字以上が含まれていてはいけません。
67	このコマンドは4D Serverで実行できません。
68	リストが必要です。
69	外部ウィンドウ参照が必要です。
70	このコマンドを2つのピクチャ引数に使用することはできません。
71	SET PRINT MARKER コマンドは印刷中のフォームのヘッダからのみ呼び出せます。
72	ポインタ配列が必要です。
73	数値配列が必要です。
74	配列サイズが一致しません。
75	No pointer on local arrays.
76	配列タイプが正しくありません。
77	変数名が不正です。
78	不正なソート引数です。
79	このコマンドはリストの描画時に実行できません。
80	クエリ引数が多すぎます。
81	フォームが見つかりません。

Tips

これらのエラーコードのうちいくつかはタイプミスによる単純なシンタックスエラーを表します。例えばコードv=0を実行するとエラー#37が発生します。これは本来v:=0と書くべきところです。このようなエラーはメソッドエディタでコードを修正することで取り除くことができます。

いくつかのエラーは単なるプログラミングエラーが原因です。例えば (DEFAULT TABLE コマンドで) デフォルトテーブルを設定せず、テーブル引数なしでADD RECORD コマンドを実行すると、エラー #5 が発生します。この場合、コマンドが対象とするテーブルがありません。このようなエラーはコードをチェックしてデフォルトテーブルを設定し忘れていないか、またはテーブル引数を忘れていないかを確認します。

いくつかのエラーは設計上のミスに基づきます。例えば他のフィールドにリレートしていないフィールドにRELATE ONEを実行すると、エラー#16が発生します。このような場合はコードが間違っていないか、またはリレーションの設定を忘れていないかをチェックします。

いくつかのエラーは、それが発生したときに、コードが停止した場所が必ずしもエラーの原因となっているとは限りません。例えばサブルーチンの中のvpFld:=Field($1;$2)行でエラー#53 (範囲外のインデックス) が発生した場合、エラーは引数として渡されたテーブルあるいはフィールド番号が正しくないことが原因です。すなわちエラーは呼び出し元のメソッドにあり、エラーが実際に発生した場所ではありません。この場合、デバッガウィンドウでコードをトレースし、コードのどの場所が本当のエラーの個所なのかを突き止 め、それをメソッドエディタで修正します。

データベースエンジンエラー (-10600 -> 4004)

この表では4Dデータベースエンジンが生成するエラーをリストしています。これらのコードはユーザによる中断、権限エラー、破損したオブジェクトなどデータベースエンジンの低レベルで発生するエラーをカバーします。

コード	説明
-10602	開かれた印刷ジョブが存在しないため、指定されたコマンドを実行できません
-10601	印刷処理中、OBJECT DUPLICATEコマンドは動作しません
-10600	このBLOBを読めません。破損しているようです
-10518	アサートに失敗しました
-10517	{folder_name} フォルダの同期に失敗しました
-10516	サーバはメンテナンス処理を実行中です。後で接続してください
-10515	4D Serverへの接続の試みが拒否されました
-10514	インストールされたライセンスの最大ユーザ数に達しました
-10513	リモートモードの4Dからはコマンド {cmd_name} を呼び出せません
-10512	このエンコーディングはサポートされていません
-10511	コンポーネントからコマンド "" を呼び出せません
-10510	コンポーネント "" をロードできません
-10509	データベース "" を開けません
-10508	プロジェクトメソッドが見つかりません
-10507	このバージョンはコンパイルされたデータベースを開くことができません
-10506	Standard Editionバージョンの制限です
-10505	クライアントとサーバのバージョン番号に互換がありません
-10504	インデックスページ番号が範囲外です (インデックスを修復する必要があります)
-10503	レコード番号が範囲外です (GOTO RECORDにおいて、あるいはデータファイルを修復する必要があります)。(注3参照)
-10502	無効なレコードストラクチャです (データファイルを修復する必要があります)
-10501	無効なインデックスページです (インデックスを修復する必要があります)
-10500	無効なレコードアドレスです (データベースを修復する必要があります)
-9999	レコードを保存するスペースがありません (注4参照)
-9998	キーが重複しています
-9997	レコード数の最大値に達しました
-9996	スタックがいっぱいです (再帰またはネストの呼び出しが多すぎます)
-9995	デモの制限に達しました
-9994	ユーザによりシリアル通信が中断されました。ユーザがCtrl-Alt-Shift (Windows) または Command-Option-Shift (Mac OS) を押しました
-9993	メニューバーが破損しています (データベースを修復する必要があります)
-9992	パスワードが間違っています
-9991	権限エラー
-9990	タイムアウトエラー
-9989	無効なストラクチャです (データベースを修復する必要があります)
-9988	フォームをロードできません。フォームまたはストラクチャが破損しています
-9987	他のレコードが既にこのレコードにリレートされています
-9986	自動削除中にレコードがロックされていました
-9985	Recursive integrity.
-9984	重複したインデックスエラーのため、トランザクションはキャンセルされました
-9983	同じ外部パッケージが2つインストールされています
-9982	そのレコードはワークステーションのセレクション中に含まれていないためロードできません
-9981	無効なフィールド名/フィールド番号がワークステーションから送信されました
-9980	ストラクチャがロックされているためファイルを作成できません
-9979	未知のユーザです
-9978	ユーザパスワードが正しくありません
-9977	セレクションが存在しません
-9976	バックアップ中です; 更新はできません
-9975	トランザクションデックスページをロードできません
-9974	レコードは既に削除されています
-9973	TRICリソースが同一ではありません
-9972	範囲外のテーブル番号がワークステーションからリクエストされました
-9971	範囲外のフィールド番号がワークステーションからリクエストされました
-9970	フィールドはインデックスされていません
-9969	無効なフィールドタイプがワークステーションからリクエストされました
-9968	無効なレコード位置番号がワークステーションからリクエストされました
-9967	レコードをロードできないため、更新できません
-9966	無効なタイプがワークステーションからリクエストされました
-9965	無効な検索定義テーブルがワークステーションから送信されました
-9964	無効なソート定義テーブルがワークステーションから送信されました
-9963	ワークステーションが無効なレコード番号をリクエストしました
-9962	サーバが終了中のためバックアップを実行できません
-9961	バックアップ処理は現在実行されていません
-9960	4D Backupがサーバにインストールされていません
-9959	他のユーザまたはプロセスによりバックアップ処理は既に開始されています
-9958	プロセスを開始できません
-9957	選択リストがロックされています
-9956	4D Clientと4D Serverのバージョンが異なります
-9955	QuickTimeがインストールされていません
-9954	カレントレコードがありません
-9953	ログファイルがありません
-9952	無効なデータセグメントヘッダです
-9951	このフィールドにはリレーションがありません
-9950	無効なデータセグメント番号です
-9949	ライセンスまたは権限エラー
-9948	モーダルダイアログが有効です
-9947	"4D Openの接続を許可する" チェックボックスが選択されていません
-9946	命名セレクションが存在しないためクリアできません
-9945	CD-ROM 4D Runtimeエラー; 書き込み処理は許可されていません
-9944	このユーザは4D Openアクセスグループに属していません
-9943	4D Connectivity Plug-insのバージョンエラー
-9942	4D Clientの待ち受けスキームがこのバージョンの4D Serverと互換がありません
-9941	未知のEX_GESTALTセレクタです
-9940	4D Extensionの初期化に失敗しました
-9939	外部ルーチンが見つかりません
-9938	カレントレコードがトリガ内で変更されました
-9937	パスワードシステムは他のユーザによりロックされています
-9936	Password extern code does not correspond to the database one
-9935	XMLファイルが妥当でないか整形式ではありません
-9934	XMLファイルが整形式ではありません
-9933	XMLファイルが無効です
-9932	XML DLLがロードされていません
-9931	この属性のインデックスが無効です
-9930	この要素にこの名前の属性はありません
-9929	この要素のインデックスが無効です
-9928	要素の名前が未知です
-9927	参照された要素はルートではありません
-9926	参照された要素が無効です
-9925	参照された要素は空です
-9924	ファイルは読み込みのみで開かなくてはなりません
-9923	属性名が無効です
-9922	属性定義中に引数値が足りません
-9921	空でないドキュメントにXML Prologを書き込もうとしています
-9920	ノードのタイプが無効です
-9919	このエンコーディングはサポートされていません
-9918	要素の名前が無効です
-9917	引数に渡された配列の型が無効です
-9916	要素が開かれていません
-9915	ドキュメント参照が無効です
-9914	内部的な障害
-9913	ネットワークの障害
-9912	HTTPの障害
-9911	パーサの障害
-9910	SOAPの障害
-9909	フォームを実行するウィンドウがありません。
-9855	無効な引数番号 5.
-9854	無効な引数番号 4.
-9853	無効な引数番号 3.
-9852	無効な引数番号 2.
-9851	無効な引数番号 1.
-9850	外部コマンドに無効なエリア引数が渡されました。
-9800	プロセスの一つがアクセス権を変更しました。
-9799	Error during preview initialization.
-9778	An error occurred while loading the dictionary
-9777	The method language does not match the application localization: {path}
-9776	Unable to create method: {path}
-9775	Unable to write method code: {path}
-9774	Unable to read method code: {path}
-9773	Unable to write method comments: {path}
-9772	Unable to read method comments: {path}
-9771	Unable to write method properties: {path}
-9770	Unable to read method properties: {path
-9769	Invalid method property: {path}
-9768	Invalid object path: {path}
-9767	Cannot update methods. One or more resources are locked.
-9766	The method is currently being edited.
-9765	One or more comments are currently being edited.
-9764	The comment is currently being edited.
-9763	The command cannot be executed in a component.
-9762	The command cannot be executed in a compiled database.
-9761	Invalid object type.
-9759	オブジェクトライブラリを開けません。
-9758	このユーザフォームは既に存在します。
-9757	このユーザフォームは存在しません。
-9756	ユーザストラクチャファイルがありません。
-9755	このユーザフォームには名前がありません。
-9754	このコマンドはダイアログウィンドウからは使用できません。
-9753	このソースフォームは存在しません。
-9752	このユーザフォームは作成できません。
-9751	ユーザはこのソースフォームにアクセスできません。
-9750	このソースフォームは編集できません。
-1	プラグインから未知のエントリポイントがリクエストされました
1001	テーブル {TableName} に読み込むカラムがありません。
1002	読み込むテーブルが正しくありません。
1003	テーブル {TableName} に書き出すカラムがありません。
1004	書き出すテーブルが正しくありません。
1006	ユーザによりプログラムが中断されました。Alt-クリック (Windows) または Option-クリック (Mac OS) が押されました。
1006	データベース {BaseName} のストラクチャを保存できません。
1007	データベース {BaseName} でデータファイルを作成できません。
1008	データベース {BaseName} のデータセグメントが不正です。
1009	メモリがいっぱいです。
1010	データベース {BaseName} のテーブル定義がロードできません。
1011	データベース {BaseName} のデータファイルを開けません。
1012	データベース {BaseName} のデータセグメントがいっぱいです。
1013	データベース {BaseName} にデータセグメントを保存できません。
1014	データベース {BaseName} のデータセグメントを読み込めません。
1015	データベース {BaseName} のデータベースヘッダが不正です。
1016	データベース {BaseName} にテーブルを作成できません。
1017	データベース {BaseName} のインデックスリストを読み込めません。
1018	データベース {BaseName} にインデックスリストを書き込めません。
1019	データベース {BaseName} のテーブル参照が不正です。
1020	データベース {BaseName} のフィールド参照が不正です。
1021	データベース {BaseName} のインデックスタイプが無効です。
1022	データベース {BaseName} のテーブル {TableName} のフィールド名が無効です。
1023	データベース名が無効です。
1024	データベース {BaseName} のストラクチャを開けません。
1025	データベース {BaseName} のストラクチャを作成できません。
1026	データベース {BaseName} のビットセレクションをロードできません。
1027	データベース {BaseName} のセットをロードできません。
1028	データベース {BaseName} のセットを更新できません。
1029	データベース {BaseName} のセットを保存できません。
1030	データベース {BaseName} のテーブル {TableName} のBLOB {BlobNum} を保存できません。
1031	データベース {BaseName} のテーブル {TableName} のBLOB {BlobNum} をロードできません。
1032	データベース {BaseName} のテーブル {TableName} のBLOB {BlobNum} を割り当てできません。
1033	データベース {BaseName} のデータビットテーブルをロードできません。
1034	データベース {BaseName} のデータビットテーブルを保存できません。
1035	データベース {BaseName} のデータビットテーブルのテーブルをロードできません。
1036	データベース {BaseName} のデータビットテーブルのテーブルを保存できません。
1037	データベース {BaseName} のデータセグメントを閉じることができません。
1038	データベース {BaseName} のデータセグメントを削除できません。
1039	データベース {BaseName} のデータセグメントを開けません。
1040	データベース {BaseName} のデータセグメントを作成できません。
1041	データセグメントにスペースを割り当てられません。
1042	データベース {BaseName} のデータセグメントのスペースを解放できません。
1043	ファイルは書き込み保護されています。
1044	データベース {BaseName} のテーブル {TableName} 中のレコード {RecNum} のフィールドにアクセスできません。
1045	データベース {BaseName} のテーブル {TableName} 中のレコード {RecNum} のフィールド定義コードが失われています。
1046	データベース {BaseName} のテーブル {TableName} 中のレコード {RecNum} を保存できません。
1047	データベース {BaseName} のテーブル {TableName} 中のレコード {RecNum} をロードできません。
1048	データベース {BaseName} のテーブル {TableName} 中にレコードを割り当てできません。
1049	データベース {BaseName} のテーブル {TableName} 中のレコード {RecNum} を更新できません。
1050	データベース {BaseName} のテーブル {TableName} 中のレコード {RecNum} のヘッダが不正です。
1051	データベース {BaseName} のテーブル {TableName} の定義を保存できません。
1052	データベース {BaseName} のテーブル {TableName} の定義を更新できません。
1053	フィールド名が既に存在します。
1055	データベース {BaseName} のテーブル {TableName} 中のレコードを保存中にインデックスを更新できません。
1056	データベース {BaseName} のテーブル {TableName} 中のレコードを保存中にBLOBを更新できません。
1057	データベース {BaseName} のテーブル {TableName} 中のレコードを削除中にBLOBを削除できません。
1058	データベース {BaseName} のテーブル {TableName} にフィールドを追加できません。
1059	メモリにテーブルを割り当てできません。
1061	メモリにレコードを割り当てできません。
1062	データベース {BaseName} のテーブル {TableName} を完全に削除できません。
1063	データベース {BaseName} のテーブル {TableName} 中のレコード {RecNum} をロックできません。
1064	データベース {BaseName} のテーブル {TableName} 中のレコード {RecNum} をロック解除できません。
1065	データベース {BaseName} のテーブル {TableName} 中のレコード {RecNum} を削除できません。
1066	データベース {BaseName} のテーブル {TableName} 中のレコード {RecNum} はロックされています。
1067	データベース {BaseName} のテーブル {TableName} のシーケンシャルサーチを完了できません。
1068	データベース {BaseName} のテーブル {TableName} のヘッダを保存できません。
1069	データベース {BaseName} のテーブル {TableName} にデータを読み込めません。
1070	データベース {BaseName} のインデックスヘッダをロードできません。
1071	データベース {BaseName} のインデックス {IndexName} のヘッダを保存できません。
1072	データベース {BaseName} のインデックス {IndexName} のページアドレスを取得できません。
1073	データベース {BaseName} のインデックス {IndexName} のページアドレスを設定できません。
1074	データベース {BaseName} のインデックス {IndexName} を破棄できません。
1075	データベース {BaseName} のインデックス {IndexName} をソートできません。
1076	データベース {BaseName} のインデックス {IndexName} のページをロードできません。
1077	データベース {BaseName} のインデックス {IndexName} のページを保存できません。
1078	データベース {BaseName} のインデックス {IndexName} にキーを挿入できません。
1079	データベース {BaseName} のインデックス {IndexName} からキーを削除できません。
1080	データベース {BaseName} のインデックス {IndexName} を完全に削除できません。
1081	データベース {BaseName} のインデックス {IndexName} のスキャンを完了できません。
1082	データベース {BaseName} のインデックス {IndexName} のソートを完了できません。
1083	データベース {BaseName} のインデックス {IndexName} のページにキーを挿入できません。
1084	データベース {BaseName} のインデックス {IndexName} のページからキーを削除できません。
1085	データベース {BaseName} のインデックスクラスタをロードできません。
1086	データベース {BaseName} のインデックスクラスタに追加できません。
1087	データベース {BaseName} から削除できません。
1088	インデックス {IndexName} が無効です。
1089	SQL syntax error (Obsolete)
1090	SQL token not found (Obsolete)
1091	実装されていません。
1092	コードを登録できません。
1093	操作中の処理がユーザによりキャンセルされました。
1094	トランザクションの衝突
1095	データベース {BaseName} の無効なテーブル名
1096	データベース {BaseName} のテーブル {TableName} がロックされています。
1097	データベース {BaseName} はロックされています。
1098	データベース {BaseName} のデータアドレスが無効です。
1099	レコードが空です。
1100	ソースフィールドが正しくありません。
1101	処理先フィールドが正しくありません。
1102	無効なリレーション名。
1103	フィールドタイプが一致しません。
1104	リレーションが空です。
1105	データベース {BaseName} からリレーションリストをロードできません。
1106	データベース {BaseName} にリレーションリストを保存できません。
1107	クエリ&ロックが完了していません。1つ以上のレコードがロックされています。
1108	無効なレコード。
1109	不正なレコードID
1110	データベース {BaseName} にすでにリレーションが存在します。
1111	データベース {BaseName} にすでにインデックスが存在します。
1112	比較演算子が正しくありません。
1113	データバッファの終了。
1114	DB4Dバージョン番号が正しくありません。
1115	重複したキー。
1116	テーブル {TableName} のレコード {RecNum} の必須入力フィールドがNULLです。
1117	テーブル {TableName} のフィールドを必須入力に設定できません。
1118	テーブル {TableName} への排他アクセスを取得できません。
1119	テーブル {TableName} の参照整合性を検証できません。
1120	参照整合性: テーブル {TableName} のレコード {RecNum} の主キーに一致するいくつかの外部キーがまだ存在します。
1121	テーブル {TableName} のセレクションの全レコードを削除できません。
1122	BLOBがNULLです。
1123	データベースコンテキストが無効です。
1124	リレート参照が無効です。
1125	テーブル {TableName} のレコード名が無効です。
1126	フィールドタイプが不正です。
1127	追加のプロパティをロードできません。
1128	追加のプロパティを保存できません。
1129	サブレコードIDが範囲外です。
1130	データベース {BaseName} のインデックス {IndexName} の名前が重複しています。
1131	データベース {BaseName} のインデックス {IndexName} の名前が無効です。
1132	インデックス {IndexName} のキー値が不正です。
1133	インデックス {IndexName} のタイプが不正です。
1134	無効なアクセス機構。
1135	アクセス機構は読み込みのみですy。
1136	NULL値は許可されません。
1137	これはNULLです。
1138	セレクションがNULLです。
1139	データベース {BaseName} は書き込み保護されています。
1140	データベース {BaseName} は閉じられようとしています。
1141	無効なトランザクション。
1142	配列の限界を超えました。
1143	配列値のクリエータが失われています。
1144	テーブル {TableName} のセレクションを構築できません。
1145	使用中のオブジェクト
1146	データファイルがストラクチャファイルと一致しません。
1147	リスナを開始できません。
1148	サーバを開始できません。
1149	リスナがありません。
1150	タスクが終了中です。
1151	無効なリクエストタグ。
1152	無効なコンテキストID。
1153	ディスク上に一時的な空きが足りません。
1154	データセットがNULLです。
1155	外部キーに一致する主キーがありません。
1156	テーブル {TableName} のフィールド {FieldName} のタイプは重複不可をサポートしません。
1157	テーブル {TableName} のフィールド {FieldName} のタイプはNULL値を許可しないをサポートしません。
1158	テーブル {TableName} の主キー定義を変更できません。
1159	テーブル {TableName} の最大レコード数に達しました。
1160	テーブル {TableName} のBLOBの最大数に達しました。
1161	インデックスが範囲外です。
1162	クエリが無効です。
1163	レコードがNULLです。
1164	オブジェクトがNULLです。
1165	このオブジェクトのオーナーが正しくありません。
1166	オブジェクトがロックされていました。
1167	オブジェクトは他のコンテキストでロックされています。
1168	リモート接続の内部エラーです。
1169	無効なテーブル番号。

1170	無効なフィールド番号。
1171	無効なデータベースID。
1172	無効な引数
1173	キャッシュの書き出しを完了できません。
1174	データの書き出しを完了できません。
1175	ストラクチャの書き出しを完了できません。
1176	データベース {BaseName} のログファイルが無効です。
1177	データベース {BaseName} のログファイルが見つかりません。
1178	ログファイル中の最後の処理が、データベース {BaseName} のそれと一致しません。
1179	ログファイルがデータベース {BaseName} と一致しません。
1180	データテーブルは削除されました。
1181	インデックス {IndexName} のキーがユニークでありません。
1182	データベース {DataBase} のログファイルを作成できません。
1183	データベース {DataBase} のログファイルに書き込みできません。
1184	データベース {DataBase} のテーブル {TableName} を削除できません。
1185	リモートデータベースを開けません。
1186	データベース {DataBase} にログファイルを統合できません。
1187	セットに対する内部演算が完了できません。
1188	配列を保存できません。
1189	配列をロードできません。
1190	シーケンス番号ヘッダをロードできません。
1191	レコードを選択できません。
1192	レコードを作成できません。
1193	データベース {DataBase} のテーブル {TableName} のセレクションを配列にできません。
1194	データベース {DataBase} のテーブル {TableName} の配列をセレクションにできません。
1195	シーケンシャルソートを完了できません。
1196	セレクションをロックできません。
1197	インデックスキーをロードできません。
1198	インデックスキーを保存できません。
1199	インデックスキーを構築できません。
1200	クエリを完了できません。
1201	クエリを解析できません。
1202	このカラムでフォーミュラを処理できません。
1203	クエリを完了できませんでした。
1204	クエリを解析できませんでした。
1205	データベース {DataBase} のテーブル {TableName} のすべての重複しない値を取得できませんでした。
1206	データベース {DataBase} のテーブル {TableName} の値配列を構築できませんでした。
1207	セレクションをロードできません。
1208	データを送信できません。
1209	リクエストの返答を受信できません。
1210	リクエストを送信できません。
1211	接続を作成できません。
1212	データベース {DataBase} でインデックス {IndexName} を素早く作成できません。
1213	重複しないインデックスキーを作成できません。
1214	データベース {DataBase} のテーブル {TableName} のセレクションをソートできません。
1215	アドレステーブルをロードできません。
1216	アドレステーブルを更新できません。
1217	アドレステーブルに新規入力ができません。
1218	アドレステーブルからフリーエントリを割り当てできません。
1219	トランザクション一時的レコードを保存できません。
1220	トランザクション一時的BLOBを保存できません。
1221	トランザクション一時的レコードをロードできません。
1222	トランザクション一時的BLOBをロードできません。
1223	トランザクションを開始できません。
1224	トランザクションをコミットできません。
1225	追加のプロパティを取得できません。
1226	追加のプロパティを設定できません。
1227	テーブル名は既に使用されています。
1228	インデックス {IndexName} からNULLキーのリストを取得できません。
1229	{IndexName}のNULLキーのリストを更新できません。
1230	インデックス {IndexName} の無効なキー。
1231	ログファイルを設定できません。
1232	コンテキストがNULLです。
1233	データベース {BaseName} をロックできません。
1234	データベース {BaseName} のテーブル {TableName} のレコード番号 {RecNum} のフィールド参照が不正です。
1235	データベース {BaseName} のテーブル {TableName} のフィールド参照が不正です。
1236	一時的なトランザクションファイルからデータを読み込めません。
1237	Cartesian Product failed
1238	セレクションをマージできません。
1239	データベースの {BaseName} のフォーマットは読み込みのみモードにアップグレードできません。
1240	不正なヘッダ。
1241	不正なチェックサム。
1243	データベース {BaseName} のデータテーブルがロードできません。
1244	データベース {BaseName} のテーブル {TableName} の外部キー制約リストが空ではありません。
1245	アドレスエントリが空ではありません。
1246	アドレスを事前に割り当てできません。
1247	データベース {BaseName} のテーブル {TableName} の新規レコードを更新できません。
1248	データベース {BaseName} のテーブル {TableName} に新規レコードを保存できません。
1249	サブレコードを保存できません。
1250	レコードを保存できません。
1251	ストラクチャオブジェクト定義をロックできません。
1252	ストラクチャオブジェクト定義をロック解除できません。
1253	リレーション番号が無効です。
1254	データベース {BaseName} のテーブル {TableName} のレコードアドレステーブルに循環参照があります。
1255	データベース {BaseName} のテーブル {TableName} のBLOBアドレステーブルに循環参照があります。
1256	データベース {BaseName} に重複したスキーマ名があります。
1257	データベース {BaseName} にスキーマを保存できません。
1258	データベース {BaseName} からスキーマを削除できません。
1259	データベース {BaseName} のスキーマを名称変更できません。
1260	データベース {BaseName} の選択されたログファイルが新しすぎます。
1261	データベース {BaseName} の選択されたログファイルが古すぎます。
1262	いくつかの DataTables に対応するテーブル定義がデータベース {BaseName} にありません。
1263	指定されたスタンプが、{TableName} テーブルのレコード# {RecNum} の現在のスタンプに該当しません。
1264	主キーが必要ですが、{TableName} テーブルに存在しません。
1265	無効なフィールドです。
1266	未知のフィールドタイプです (4Dのバージョンが古い可能性があります)。
1267	スタックオーバーフローです。
1268	データベース {BaseName} の DataTable を再構築できません。
1269	DataTable が見つかりません。
1270	失われたストラクチャーを再構築できません。
1271	無効なRESTリクエストハンドラーです。
1272	データベース {BaseName} の {tableName} テーブル中でいくつかのレコードがまだロックされています。
1273	データベース {BaseName} の {tableName} テーブルを削除できません。
1274	データベース {BaseName} のカレントジャーナルファイルを利用できません。データ保護のため書き込み処理を停止します。できるだけ早く管理者に連絡してください。
1275	このクエリのJavaScript文の先頭で "(" が失われています。
1276	Datastream に無効なヘッダーが含まれています。
1277	データベースジャーナルの統合が entry# {p1} で失敗しました。
1278	クエリ中でJavaScriptコードは許可されていません。
1300	無効なセレクションタイプ。
1301	配列が大きすぎます。
1302	配列サイズが一致しません。
1303	無効なセレクションID。
1304	Invalid Selection Part
4001	プラグインから要求されたテーブル番号が無効です
4002	プラグインから要求されたレコード番号が無効です
4003	プラウインから要求されたフィールド番号が無効です
4004	プラグインがテーブルのカレントレコードにアクセスを試みましたが、カレントレコードがありません

注:

1. リストされたエラーのうちいくつかは深刻なエラーを反映します (例えば-10502 無効なレコードストラクチャです (データファイルを修復する必要があります))。その他のエラーは通常の処理で発生し、ON ERR CALLでインストールするプロジェクトメソッドで管理できます。例えばアプリケーションにおいて重複不可プロパティが設定されたインデックス付きフィールドに重複した値が作成される可能性がある場合、通常エラー-9998 キーが重複していますを処理します。
2. いくつかのエラーは4Dランゲージレベルで発生することはありません。これらはデータベースエンジンルーチンや、4D Backupおよび4D Openの利用時に低レベルで発生および処理されます。
3. エラー-10503 レコード番号が範囲外です は通常、(例えばGOTO RECORDコマンドなどの) コードが存在しないレコードにアクセスしようとしたときに発生します。まれなケースとしては、データベースの修復が必要となることを意味することもあります。
4. エラー-9999 レコードを保存するスペースがありませんはデータベースのデータファイルがいっぱいか、ボリュームがいっぱいのときに発生します。あるいはデータファイルまたはボリュームがロックされている場合にも発生します。

SQLエンジンエラー (1001 -> 3018)

4DのSQLエンジンは以下にリストするようなエラーを返します。これらのエラーはON ERR CALLでインストールされるエラー処理メソッドでとらえることができ、GET LAST ERROR STACKコマンドで検証できます。

一般的なエラー

1001 INVALID ARGUMENT
1002 INVALID INTERNAL STATE
1003 SQL SERVER IS NOT RUNNING
1004 Access denied
1005 FAILED TO LOCK SYNCHRONIZATION PRIMITIVE
1006 FAILED TO UNLOCK SYNCHRONIZATION PRIMITIVE
1007 SQL SERVER IS NOT AVAILABLE
1008 COMPONENT BRIDGE IS NOT AVAILABLE
1009 REMOTE SQL SERVER IS NOT AVAILABLE
1010 EXECUTION INTERRUPTED BY USER

セマンティックエラー

1101 Table '{key1}' does not exist in the database.
1102 Column '{key1}' does not exist.
1103 Table '{key1}' is not declared in the FROM clause.
1104 Column name reference '{key1}' is ambiguous.
1105 Table alias '{key1}' is the same as table name.
1106 Duplicate table alias - '{key1}'.
1107 Duplicate table in the FROM clause - '{key1}'.
1108 Operation {key1} {key2} {key3} is not type safe.
1109 Invalid ORDER BY index - {key1}.
1110 Function {key1} expects one parameter, not {key2}.
1111 Parameter {key1} of type {key2} in function call {key3} is not implicitly convertible to {key4}.
1112 Unknown function - {key1}.
1113 Division by zero.
1114 Sorting by indexed item in the SELECT list is not allowed - ORDER BY item {key1}.
1115 DISTINCT NOT ALLOWED
1116 Nested aggregate functions are not allowed in the aggregate function {key1}.
1117 Column function is not allowed.
1118 Cannot mix column and scalar operations.
1119 Invalid GROUP BY index - {key1}.
1120 GROUP BY index is not allowed.
1121 GROUP BY is not allowed with 'SELECT * FROM ...'.
1122 HAVING is not an aggregate expression.
1123 Column '{key1}' is not a grouping column and cannot be used in the ORDER BY clause.
1124 Cannot mix {key1} and {key2} types in the IN predicate.
1125 Escape sequence '{key1}' in the LIKE predicate is too long. It must be exactly one character.
1126 Bad escape character - '{key1}'.
1127 Unknown escape sequence - '{key1}'.
1128 Column references from more than one query in aggregate function {key1} are not allowed.
1129 Scalar item in the SELECT list is not allowed when GROUP BY clause is present.
1130 Sub-query produces more than one column.
1131 Subquery must return one row at the most but instead it returns {key1}.
1132 INSERT value count {key1} does not match column count {key2}.
1133 Duplicate column reference in the INSERT list - '{key1}'.
1134 Column '{key1}' does not allow NULL values.
1135 Duplicate column reference in the UPDATE list - '{key1}'.
1136 Table '{key1}' already exists.
1137 Duplicate column '{key1}' in the CREATE TABLE command.
1138 DUPLICATE COLUMN IN COLUMN LIST
1139 More than one primary key is not allowed.
1140 Ambiguous foreign key name - '{key1}'.
1141 Column count {key1} in the child table does not match column count {key2} in the parent table of the foreign key definition.
1142 Column type mismatch in the foreign key definition. Cannot relate {key1} in child table to {key2} in parent table.
1143 Failed to find matching column in parent table for '{key1}' column in child table.
1144 UPDATE and DELETE constraints must be the same.
1145 FOREIGN KEY DOES NOT EXIST
1146 Invalid LIMIT value in SELECT command - {key1}.
1147 Invalid OFFSET value in SELECT command - {key1}.
1148 Primary key does not exist in table '{key1}'.
1149 FAILED TO CREATE FOREIGN KEY
1150 Column '{key1}' is not part of a primary key.
1151 FIELD IS NOT UPDATEABLE
1152 FOUND VIEW COLUMN
1153 Bad data type length '{key1}'.
1154 EXPECTED EXECUTE IMMEDIATE COMMAND
1155 INDEX ALREADY EXISTS
1156 Auto-increment option is not allowed for column '{key1}' of type {key2}.
1157 SCHEMA ALREADY EXISTS
1158 SCHEMA DOES NOT EXIST
1159 Cannot drop system schema
1160 CHARACTER ENCODING NOT ALLOWED

実装エラー

1203 FUNCTIONALITY IS NOT IMPLEMENTED
1204 Failed to create record {key1}.
1205 Failed to update field '{key1}'.
1206 Failed to delete record '{key1}'.
1207 NO MORE JOIN SEEDS POSSIBLE
1208 FAILED TO CREATE TABLE
1209 FAILED TO DROP TABLE
1210 CANT BUILD BTREE FOR ZERO RECORDS
1211 COMMAND COUNT GREATER THAN ALLOWED
1212 FAILED TO CREATE DATABASE
1213 FAILED TO DROP COLUMN
1214 VALUE IS OUT OF BOUNDS
1215 FAILED TO STOP SQL_SERVER
1216 FAILED TO LOCALIZE
1217 Failed to lock table for reading.
1218 FAILED TO LOCK TABLE FOR WRITING
1219 TABLE STRUCTURE STAMP CHANGED
1220 FAILED TO LOAD RECORD
1221 FAILED TO LOCK RECORD FOR WRITING
1222 FAILED TO PUT SQL LOCK ON A TABLE
1223 FAILED TO RETAIN COOPERATIVE TASK
1224 FAILED TO LOAD INFILE

解析エラー

1301 PARSING FAILED

ランタイムランゲージアクセスエラー

1401 COMMAND NOT SPECIFIED
1402 ALREADY LOGGED IN
1403 SESSION DOES NOT EXIST
1404 UNKNOWN BIND ENTITY
1405 INCOMPATIBLE BIND ENTITIES
1406 REQUEST RESULT NOT AVAILABLE
1407 BINDING LOAD FAILED
1408 COULD NOT RECOVER FROM PREVIOUS ERRORS
1409 NO OPEN STATEMENT
1410 RESULT EOF
1411 BOUND VALUE IS NULL
1412 STATEMENT ALREADY OPENED
1413 FAILED TO GET PARAMETER VALUE
1414 INCOMPATIBLE PARAMETER ENTITIES
1415 Query parameter is either not allowed or was not provided.
1416 COLUMN REFERENCE PARAMETERS FROM DIFFERENT TABLES
1417 EMPTY STATEMENT
1418 FAILED TO UPDATE VARIABLE
1419 FAILED TO GET TABLE REFERENCE
1420 FAILED TO GET TABLE CONTEXT
1421 COLUMNS NOT ALLOWED
1422 INVALID COMMAND COUNT
1423 INTO CLAUSE NOT ALLOWED
1424 EXECUTE IMMEDIATE NOT ALLOWED
1425 ARRAY NOT ALLOWED IN EXECUTE IMMEDIATE
1426 COLUMN NOT ALLOWED IN EXECUTE IMMEDIATE
1427 NESTED BEGIN END SQL NOT ALLOWED
1428 RESULT IS NOT A SELECTION
1429 INTO ITEM IS NOT A VARIABLE
1430 VARIABLE WAS NOT FOUND
1431 PTR OF PTR NOT ALLOWED
1432 POINTER OF UNKNOWN TYPE

日付解析エラー

1501 SEPARATOR_EXPECTED
1502 FAILED TO PARSE DAY OF MONTH
1503 FAILED TO PARSE MONTH
1504 FAILED TO PARSE YEAR
1505 FAILED TO PARSE HOUR
1506 FAILED TO PARSE MINUTE
1507 FAILED TO PARSE SECOND
1508 FAILED TO PARSE MILLISECOND
1509 INVALID AM PM USAGE
1510 FAILED TO PARSE TIME ZONE
1511 UNEXPECTED CHARACTER
1512 Failed to parse timestamp.
1513 Failed to parse duration.
1551 FAILED TO PARSE DATE FORMAT

Lexer errors

1601 NULL INPUT STRING
1602 NON TERMINATED STRING
1603 NON TERMINATED COMMENT
1604 INVALID NUMBER
1605 UNKNOWN START OF TOKEN
1606 NON TERMINATED NAME/* closing ']' is missing
1607 NO VALID TOKENS

Lexer エラー

1601 NULL INPUT STRING
1602 NON TERMINATED STRING
1603 NON TERMINATED COMMENT
1604 INVALID NUMBER
1605 UNKNOWN START OF TOKEN
1606 NON TERMINATED NAME/* closing ']' is missing
1607 NO VALID TOKENS

エラースタックの実行エラー - state errors that follow direct errors

1801 Failed to execute SELECT command.
1802 Failed to execute INSERT command.
1803 Failed to execute DELETE command.
1804 Failed to execute UPDATE command.
1805 Failed to execute CREATE TABLE command.
1806 Failed to execute DROP TABLE command.
1807 Failed to execute CREATE DATABASE command.
1808 Failed to execute ALTER TABLE command.
1809 Failed to execute CREATE INDEX command.
1810 Failed to execute DROP INDEX command.
1811 Failed to execute LOCK TABLE command.
1812 Failed to execute TRANSACTION command.
1813 Failed to execute WHERE clause.
1814 Failed to execute GROUP BY clause.
1815 Failed to execute HAVING clause.
1816 Failed to aggregate.
1817 Failed to execute DISTINCT.
1818 Failed to execute ORDER BY clause.
1819 Failed to build DB4D query.
1820 Failed to calculate comparison predicate.
1821 Failed to execute subquery.
1822 Failed to calculate BETWEEN predicate.
1823 Failed to calculate IN predicate.
1824 Failed to calculate ALL/ANY predicate.
1825 Failed to calculate LIKE predicate.
1826 Failed to calculate EXISTS predicate.
1827 Failed to calculate NULL predicate.
1828 Failed to perform arithmetic operation.
1829 Failed to calculate function call '{key1}'.
1830 Failed to calculate case expression.
1831 Failed to calculate function parameter '{key1}'.
1832 Failed to calculate 4D function call.
1833 Failed to sort while executing ORDER BY clause.
1834 Failed to calculate record hash.
1835 Failed to compare records.
1836 Failed to calculate INSERT value {key1}.
1837 DB4D QUERY FAILED
1838 FAILED TO EXECUTE ALTER SCHEMA COMMAND
1839 FAILED TO EXECUTE GRANT COMMAND

キャッシュエラー

2000 CACHEABLE NOT INITIALIZED
2001 VALUE ALREADY CACHED
2002 CACHED VALUE NOT FOUND
2003 CACHE IS FULL
2004 CACHING IS NOT POSSIBLE

プロトコルエラー

3000 HEADER NOT FOUND
3001 UNKNOWN COMMAND
3002 ALREADY LOGGED IN
3003 NOT LOGGED IN
3004 UNKNOWN OUTPUT MODE
3005 INVALID STATEMENT ID
3006 UNKNOWN DATA TYPE
3007 STILL LOGGED IN
3008 SOCKET READ ERROR
3009 SOCKET WRITE ERROR
3010 BASE64 DECODING ERROR
3011 SESSION TIMEOUT
3012 FETCH TIMESTAMP ALREADY EXISTS
3013 BASE64 ENCODING ERROR
3014 INVALID HEADER TERMINATOR
3015 INVALID SESSION TICKET
3016 HEADER TOO LONG
3017 INVALID AGENT SIGNATURE
3018 UNEXPECTED HEADER VALUE

ネットワークエラー (-10051 -> -10001)

以下の表はネットワーク接続時に発生するエラーについて説明しています。

コード	説明
-10051	Get/SetOptionの値が正しくありません。
-10050	不明なGet/SetOptionオプションです。
-10033	読み込みサイクル中の正しくないデータサイズです。
-10031	読み込みサイクル中に非同期化が発生しました。
-10030	書き込みサイクル中に非同期化が発生しました。
-10021	OP Find 4D Serverを使用中にサーバが見つかりませんでした。
-10020	OP Select 4D Serverでサーバが選択されていません。
-10003	接続パラメタが正しくありません。
-10002	このプロセスへの接続が中断されました。
-10001	データベースへの接続が中断されました。

バックアップマネージャエラー (1401 -> 1421)

以下の表では4Dのバックアップや復元モジュールで生成されるエラーコードをリストします。
これらのエラーはON ERR CALLコマンドでインストールされたメソッドで取得できます。

コード	説明
1401	バックアップ試行の最大回数に達しました。自動バックアップスケジューラは一時的に無効になります。
1403	ログファイルがありません。
1404	このプロセスでトランザクションが開かれています。
1405	コンカレントプロセスのトランザクション終了最大タイムアウトに達しました。
1406	バックアップがユーザによりキャンセルされました。
1407	保存先フォルダが無効です。
1408	ログファイルバックアップ中にエラーがありました。
1409	バックアップ中にエラーがありました。
1410	検証するバックアップファイルが見つかりません。
1411	バックアップファイル検証中にエラーがありました。
1412	検証するログバックアップファイルが見つかりません。
1413	ログバックアップファイル検証中にエラーがありました。
1414	このコマンドは4D Serverのみで実行できます。
1415	ログファイルをバックアップできません; 重要な操作を実行中です。
1416	このログファイルは開かれているデータベースに対応していません。
1417	ログの統合処理が実行中です。バックアップを起動できません。
1420	ロックされたレコードを検知したため統合がアボートされました。
1421	このコマンドはクライアント/サーバ環境で実行できません。

- エラー1408と1409は通常、バックアップするファイルの読み込みエラーまたはファイルバックアップ中の書き込みエラーで発生します。
- エラー1411と1413はアーカイブの検証中に発生します。

これらのエラーが発生するときは、まずディスクの空き容量や読み書きアクセス権をチェックします。

OSファイルマネージャエラー (-124 -> -33)

以下の表はオペレーティングシステムのファイルマネージャが返すコードをリストしています。これらのコードは例えばシステムドキュメントコマンドを使用している際に返されます。このリスト中、"ファイル"とはディスク上のドキュメントを指します。データベースストラクチャのファイルではありません。

コード	説明
-124	接続解除された共有ボリュームにアクセスしようとしました。
-121	アクセスパスを作成できません。
-120	存在しないディレクトリを指定するパス名を使用してファイルにアクセスしようとしました。
-84	ディスクにハードウェアの問題があります (インストールやフォーマットが正しくない等)。
-64	ディスクにハードウェアの問題があります (インストールやフォーマットが正しくない等)。
-61	読み/書き権限が書き込みを許可していません。
-60	マスタディレクトリブロックが正しくありません。ディスクが損傷しています。
-58	外部ファイルシステムのエラー。
-57	非Macintoshディスクで作業しようとしました。
-54	ロックされたファイルを書き込みのために開こうとしました。
-53	ボリュームがオンラインではありません。
-52	内部的なファイルマネージャエラー (ファイルマーカの位置を失いました)。
-51	無効なドキュメント参照でドキュメントにアクセスしようとしました。
-49	すでに書き込み権限で開かれたファイルを開こうとしました。
-48	既に削除したファイルの名前で、ファイルの名称を変更しようとしました。
-47	削除されたファイルにアクセスしようとしました。
-46	アプリケーションによりボリュームがロックされています。
-45	ファイルがロックされています。
-44	ハードウェア設定によりボリュームがロックされています。
-43	ファイルが見つかりません。
-42	同時に開いたファイルが多すぎます。
-41	ディスク上のファイルを開くための十分なメモリがありません。
-40	ファイルの先頭よりも前に移動しようとしました。
-39	読み込み処理中に論理的なEnd Of File (EOF) に達しました。
-38	開かれていないファイルを読み込みまたは書き込みしようとしました。
-37	ファイルまたはボリューム名が正しくありません。
-36	I/Oエラーです。おそらくディスク上のブロックに障害があります。
-35	指定されたボリュームは存在しません。
-34	ディスクがいっぱいです。ディスクの空き容量がありません。
-33	ファイルディレクトリがいっぱいです。ディスク上に新しいファイルを作成できません。

OSメモリマネージャエラー (-117 -> -108)

以下の表はオペレーティングシステムのメモリマネージャから返されるエラーをリストしています。

コード	説明
-117	内部的なメモリの問題。おそらくメモリが論理的に壊れています。すぐに終了し、マシンを再起動してからデータベースを開いてください。
-111	内部的なメモリの問題。おそらくメモリが論理的に壊れています。すぐに終了し、マシンを再起動してからデータベースを開いてください。(*)
-109	内部的なメモリの問題。おそらくメモリが論理的に壊れています。すぐに終了し、マシンを再起動してからデータベースを開いてください。
-108	処理を行うのに十分なメモリがありません。4Dアプリケーションにより多くのメモリを与えてください。

Tip: 大量のデータを保持することのできるオブジェクト、たとえば大きな配列やBLOB、ピクチャ、セットを扱うときは、ON ERR CALLでエラー処理メソッドをインストールし、エラー-108をテストします。

(*) エラー -111 はBLOBの範囲外から値を読み込もうとしたときにも発生します。この場合、エラーは小さなものであり、ワーキングセッションを終了する必要はありません。BLOBコマンドに渡すオフセットを修正してください。

OSプリントマネージャエラー (-8192 -> -1)

以下の表はオペレーティングシステムのプリントマネージャから返されるエラーをリストしています。
これらのコードはプリント中に返されます。

コード	説明
-8192	LaserWriter タイムアウト
-8151	プリンタが異なるドライバのバージョンで初期化されています。
-8150	LaserWriterが選択されていません。
-4101	プリンタが使用中か接続されていません。
-4100	プリンタとの接続が中断されました。
-193	リソースファイルが見つかりません。
-128	プリントがユーザにより中断されました。
-27	プリンタとの接続を開くまたは閉じる際に問題があります。
-1	プリントするファイルの保存に問題があります。

OSリソースマネージャエラー (-196 -> -1)

以下の表はオペレーティングシステムのリソースマネージャから返されるエラーをリストしています。

コード	説明
-196	リソースを削除できません。
-194	リソースを追加できません。
-193	リソースマップが破損しています (ファイルを修復する必要があります)。
-192	リソースが見つかりません。
-1	リソースファイルを開けません。

SANE NaNエラー (1 -> 255)

以下の表はオペレーティングシステムから返されるNaNエラーをリストしています。NaNは Standard Apple Numeric Environment (SANE) で"Not a Number"を意味するものです。NaNは処理がSANEのスコープ範囲外の結果を生成する際に現れます。

コード	説明
1	不正な平方根です。
2	不正な加算です。
4	不正な割り算です。
8	不正な乗算です。
9	不正な割り算の余りです。
17	無効なASCII 文字列を変換しようとしています。
20	Comp型の数値を浮動小数に変換しようとしています。
21	0コードでNaNを作成しようとしています。
33	三角関数の引数が不正です。
34	逆三角関数の引数が不正です。
36	対数関数の引数が不正です。
37	xi または xy 関数の引数が不正です。
38	財務関数の引数が不正です。
255	記憶領域が初期化されていません。

OSサウンドマネージャエラー (-209 -> -203)

以下の表はオペレーティングシステムのサウンドマネージャから返されるエラーをリストしています。

コード	説明
-209	サウンドチャンネルが使用中です。
-207	サウンドを実行するのに十分なメモリがありません。
-206	サウンドリソースのフォーマットが間違っています。
-205	サウンドチャンネルが論理的に壊れています。
-204	サウンドリソースをロードできません。
-203	サウンドコマンドが多すぎます。

OSシリアルポートマネージャエラー (-28)

以下の表はオペレーティングシステムのシリアルポートマネージャから返されるエラーをリストしています。

Code	説明
-28	開かれたシリアルポートがありません。

Mac OSシステムエラー (4 -> 28)

以下の表はMac OSシステムエラーをリストしています。通常これらのエラーから復帰することはできません。

コード	説明
4	0による割り算
15	セグメントロードエラー: 4Dは自身のコードセグメントのロードに失敗しました。4Dにより多くのメモリを割り当てる必要があります。
17 から 24	システムパッケージが失われています。システムディレクトリが正しくインストールされているかを確認してください。
25	メモリが足りません。4Dにより多くのメモリを割り当てる必要があります。
28	スタックがアプリケーションヒープに移動されました。4Dにより多くのメモリを割り当てる必要があります。

文字コード

- Unicodeコード
- ASCIIコード
- ファンクションキーコード

Unicodeコード

4D バージョン11で作成されたデータベースでは、ランゲージ並びにデータベースエンジンはUnicode文字をネイティブに格納及び扱います。

これにより4Dアプリケーションの国際化が容易になります。Unicodeは標準化された統合文字セットであり、実質的に世界すべての言語を扱うことができます。文字セットは文字と数値の対応表です。例えば"a"->97, "b"->98, "5"->53, "oe"->207のように対応させます

ASCIIにおいては基本的な数値は1から127に収まっています。Unicodeでは上限が65,000を超えており、ほとんどすべての言語のすべての文字を表現できます。

Unicodeの数値を表現する方法は複数あります: UTF-16は文字を16-bit整数でコード化します。UTF-32は32-bit整数を、UTF-8は8-bit整数を使用します。4Dは主にUTF-16を使用します (WindowsやMac OSと同様)。インターネットや通信関連の用途では、データ量及び共通文字 (a-z,0-9)の可読性から、4DはUTF-8を使用します。

Unicode標準に関する詳細は例えば以下のページを参照してください:
http://en.wikipedia.org/wiki/Unicode
Unicodeコードのリスト:
http://en.wikipedia.org/wiki/List_of_Unicode_characters#See_also

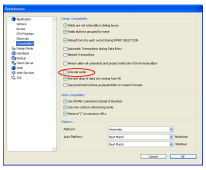
警告: 4D v11のUnicodeにおいて、以下の文字コードは予約されており、テキストに含まれていてはいけません:
0
65534 (FFFE)
65535 (FFFF)

互換性メモ: 4D v11より前に作成されたデータベースは、ASCII互換モードで動作できます。詳細はの節を参照してください。

ASCIIコード

互換性メモ: 4Dは2つの文字セット、UnicodeまたはASCIIのいずれかで動作します。Unicodeは標準モードで、バージョン11の4Dより新規データベースで使用されます。ASCIIモードは以前のバージョンの4Dで作成されたデータベースの互換性のために保持されているモードです。このモードは**ASCII互換モード**と呼ばれます。

Get database parameterとSET DATABASE PARAMETERコマンドのUnicode Modeセレクタ、または環境設定のアプリケーション/互換ページにある**Unicodeモード**オプションで、変換されたデータベースにUnicodeモードを適用できます。



Design Compatibility
- ☑ Fields are not enterable in dialog boxes
- ☑ Radio buttons grouped by name
- ☑ Reload form for each record during PRINT SELECTION
- ☐ Automatic Transactions during Data Entry
- ☐ Nested Transactions
- ☐ Always allow all commands and project methods in the Formula editor
- ☐ Unicode mode
- ☑ Prevent drop of data not coming from 4D
- ☐ Use period and comma as placeholders in numeric formats

Web Compatibility
- ☑ Use 4DVAR Comments instead of Brackets
- ☑ Use new context referencing mode
- ☑ Remove "/" on unknown URLs

Platform
- Platform: Automatic
- Auto Platform: Best Match (Macintosh)
- Best Match (Windows)

ほとんどの場合、この設定によってアプリケーションの初期機能が影響を受けることはありません。4Dはすべての必要な文字変換を内部で処理します。更に、多くの共通な文字 (a-z、0-9など) は、UnicodeとASCII(WindowsとMac OS)の両方において同じ値(1から127) が使用されます。

しかし、特に文字列に対して機能するコマンドを使用するランゲージステートメントでは、注意が必要です。例を挙げると、ステートメントChar(200)は、ASCIIとUnicodeで同じ値を返しません。このマニュアルでは、関連する各コマンドのUnicodeモードとASCII互換モード間の機能の違いについて説明します。

日本語を扱う際の注意: 4Dで日本語を使用する場合、Unicodeモードを使用することを強く推奨します。文字列を扱ういくつかのコマンドは、データベースがASCII 互換モードで動作していても、文字列処理にUnicodeのICUライブラリを使用します。この場合の動作は以前のバージョンと同じにならない場合があります。

Note: このモードはデータベースごとに適用されます。従ってUnicodeデータベースで非Unicodeコンポーネント（またはその逆）を使用できます。

ASCIIコードと4Dについて理解する

データベースがASCII互換モードで開かれるとき、MacintoshとWindows両方で、内部データベースエンジンと4DランゲージはMacintoshの拡張ASCIIセットを使用します。キーボードを使用してデータを入力する際 (レコードの追加、メソッドの編集など)、4Dは内部的なAltura ASCII変換スキームを使用してキーボードから入力される (Windowsセットで表現された) 文字をMacintoshセットに変換します。例えば"é"を入力するにはALT+0233とタイプします。4DはASCIIコード142をレコードに格納します。検索を行う際などは検索エディタに同様に探したい文字を打ち込むので、これはエンドユーザに対して透過的です。つまりタイプした値 (ALT+0233) はここでもASCIIコード142に翻訳され、値を検索できます。

メソッドエディタにALT+0233とタイプした場合も同様に動作します。しかしASCIIコードを使用して文字を探すためには、文字のMacintosh ASCIIコードを使用してください。

例えば:

```
QUERY(...;[MyFile]MyField="é") ` é は Alt+0233
```

これは以下のコードと同じです:

```
QUERY(...;[MyFile]MyField=Char(142)) ` é は ASCII 142
```

ASCIIコード表

- 0から127の標準ASCIIコードはWindowsとMacintoshで共通です。
- 128から255のASCIIコードはWindowsとMacintoshで異なります。プラットフォーム間の差異を保守するために、Windows版の4Dは、4D環境に文字が入力 (データ入力、編集/ペーストなど) されると自動でASCIIコードがWindowsからMacintoshのASCIIマップに変換され、4D環境から文字が出力 (編集/カットやコピー、書き出し等) される際にはMacintoshからWindows ASCIIマップに変換されます。

ASCIIコード 0 ～ 63

ASCII		Macintosh or Windows	ASCII		Macintosh or Windows
Dec	Hex	Result	Dec	Hex	Result
0	0	NUL	16	10	DLE
1	1	SOH	17	11	DC1
2	2	STX	18	12	DC2
3	3	ETX	19	13	DC3
4	4	EOT	20	14	DC4
5	5	ENQ	21	15	NAK
6	6	ACK	22	16	SYN
7	7	BEL	23	17	ETB
8	8	BS	24	18	CAN
9	9	HT	25	19	EM
10	A	LF	26	1A	SUB
11	B	VT	27	1B	ESC
12	C	FF	28	1C	FS
13	D	CR	29	1D	GS
14	E	SO	30	1E	RS
15	F	SI	31	1F	US

ASCII		Macintosh or Windows	ASCII		Macintosh or Windows
Dec	Hex	Result	Dec	Hex	Result
32	20	sp	48	30	0
33	21	!	49	31	1
34	22	"	50	32	2
35	23	#	51	33	3
36	24	$	52	34	4
37	25	%	53	35	5
38	26	&	54	36	6
39	27	'	55	37	7
40	28	(	56	38	8
41	29	)	57	39	9
42	2A	*	58	3A	:
43	2B	+	59	3B	;
44	2C	,	60	3C	<
45	2D	-	61	3D	=
46	2E	.	62	3E	>
47	2F	/	63	3F	?

ASCIIコード 64 ～ 127

ASCII		Macintosh or Windows	ASCII		Macintosh or Windows
Dec	Hex	Result	Dec	Hex	Result
64	40	@	80	50	P
65	41	A	81	51	Q
66	42	B	82	52	R
67	43	C	83	53	S
68	44	D	84	54	T
69	45	E	85	55	U
70	46	F	86	56	V
71	47	G	87	57	W
72	48	H	88	58	X
73	49	I	89	59	Y
74	4A	J	90	5A	Z
75	4B	K	91	5B	[
76	4C	L	92	5C	\
77	4D	M	93	5D	]
78	4E	N	94	5E	^
79	4F	O	95	5F	_

ASCII		Macintosh or Windows	ASCII		Macintosh or Windows
Dec	Hex	Result	Dec	Hex	Result
96	60	`	112	70	p
97	61	a	113	71	q
98	62	b	114	72	r
99	63	c	115	73	s
100	64	d	116	74	t
101	65	e	117	75	u
102	66	f	118	76	v
103	67	g	119	77	w
104	68	h	120	78	x
105	69	i	121	79	y
106	6A	j	122	7A	z
107	6B	k	123	7B	{
108	6C	l	124	7C	|
109	6D	m	125	7D	}
110	6E	n	126	7E	~
111	6F	o	127	7F	DEL

ASCIIコード 128 ～ 191

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
128	80	Ä	Option-u, Shift-a	Ä	Alt+0196
129	81	Å	Option-Shift-a	Å	Alt+0197
130	82	Ç	Option-Shift-c	Ç	Alt+0199
131	83	É	Option-e, Shift-e	É	Alt+0201
132	84	Ñ	Option-n, Shift-n	Ñ	Alt+0209
133	85	Ö	Option-u, Shift-o	Ö	Alt+0214
134	86	Ü	Option-u, Shift-u	Ü	Alt+0220
135	87	á	Option-e, a	á	Alt+0225
136	88	à	Option-`, a	à	Alt+0224
137	89	â	Option-i, a	â	Alt+0226
138	8A	ä	Option-u, a	ä	Alt+0228
139	8B	ã	Option-n, a	ã	Alt+0227
140	8C	å	Option-a	å	Alt+0229
141	8D	ç	Option-c	ç	Alt+0231
142	8E	é	Option-e, e	é	Alt+0233
143	8F	è	Option-`, e	è	Alt+0232

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
144	90	ê	Option-i, e	ê	Alt+0234
145	91	ë	Option-u, e	ë	Alt+0235
146	92	í	Option-e, i	í	Alt+0237
147	93	ì	Option-`, i	ì	Alt+0236
148	94	î	Option-i, i	î	Alt+0238
149	95	ï	Option-u, i	ï	Alt+0239
150	96	ñ	Option-n, n	ñ	Alt+0241
151	97	ó	Option-e, o	ó	Alt+0243
152	98	ò	Option-`, o	ò	Alt+0242
153	99	ô	Option-i, o	ô	Alt+0244
154	9A	ö	Option-u, o	ö	Alt+0246
155	9B	õ	Option-n, o	õ	Alt+0245
156	9C	ú	Option-e, u	ú	Alt+0250
157	9D	ù	Option-`, u	ù	Alt+0249
158	9E	û	Option-i, u	û	Alt+0251
159	9F	ü	Option-u, u	ü	Alt+0252

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
160	A0	†	Shift-Option-7		
161	A1	°	Shift-Option-7	°	Alt+0176
162	A2	¢	Option-4	¢	Alt+0162
163	A3	£	Option-3	£	Alt+0163
164	A4	§	Option-6	§	Alt+0167
165	A5	•	Option-8		
166	A6	¶	Option-7	¶	Alt+0182
167	A7	ß	Option-s	ß	Alt+0223
168	A8	®	Option-r	®	Alt+0174
169	A9	©	Option-g	©	Alt+0169
170	AA	™	Option-2	™	Alt-0153
171	AB	´	Shift-Option-e	´	Alt+0145
172	AC	¨	Shift-Option-u	¨	Alt+0168
173	AD	≠	Option-=		
174	AE	Æ	Shift-Option-'	Æ	Alt+0198
175	AF	Ø	Shift-Option-o	Ø	Alt+0216

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
176	B0	∞	Option-5		
177	B1	±	Shift-Option-=	±	Alt+0177
178	B2	≤	Option-,		
179	B3	≥	Option-. (period)		
180	B4	¥	Option-y	¥	Alt+0165
181	B5	µ	Option-m	µ	Alt+0181
182	B6	∂	Option-d		
183	B7	∑	Option-w		
184	B8	∏	Shift-Option-p		
185	B9	π	Option-p		
186	BA	∫	Option-b		
187	BB	ª	Option-9	ª	Alt+0170
188	BC	º	Option-0 (zero)	º	Alt+0186
189	BD	Ω	Option-z		
190	BE	æ	Option-'	æ	Alt+0230
191	BF	ø	Option-o	ø	Alt+0248

ASCIIコード 192 ～ 255

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
192	C0	¿	Shift-Option-?	¿	Alt+0191
193	C1	¡	Option-1	¡	Alt+0161
194	C2	¬	Option-l (L)	¬	Alt+0172
195	C3	√	Option-v		
196	C4	ƒ	Option-f		
197	C5	≈	Option-x		
198	C6	∆	Option-j		
199	C7	«	Option-\	«	Alt+0171
200	C8	»	Shift-Option-\	»	Alt+0187
201	C9	…	Option-;	…	Alt+0133
202	CA	(space)	Spacebar	(space)	Alt+0160
203	CB	À	Option-`, Shift-a	À	Alt+0192
204	CC	Ã	Option-n, Shift-a	Ã	Alt+0195
205	CD	Õ	Option-n, Shift-o	Õ	Alt+0213
206	CE	Œ	Shift-Option-q		
207	CF	œ	Option-q		

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
208	D0	–	Option--(dash)		
209	D1	—	Shift-Option-(dash)		
210	D2	“	Option-[	“	Alt+0147
211	D3	”	Shift-Option-[	”	Alt+0148
212	D4	‘	Option-]	‘	Alt+0145
213	D5	’	Shift-Option-]	’	Alt+0146
214	D6	÷	Option-/	÷	Alt+0247
215	D7	◊	Shift-Option-v		
216	D8	ÿ	Option-u, y	ÿ	Alt+0255
217	D9	Ÿ	Option-u, Shift-y		
218	DA	⁄	Shift-Option-1		
219	DB	€	Shift-Option-2	¤	Alt+0164
220	DC	‹	Shift-Option-3		
221	DD	›	Shift-Option-4		
222	DE	ﬁ	Shift-Option-5		
223	DF	ﬂ	Shift-Option-6		

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
224	E0	‡	Shift-Option-7		
225	E1	·	Shift-Option-9	·	Alt+0183
226	E2	‚	Shift-Option-0		
227	E3	„	Shift-Option-w		
228	E4	‰	Shift-Option-r		
229	E5	Â	Option-i, Shift-a	Â	Alt+0194
230	E6	Ê	Option-i, Shift-e	Ê	Alt+0202
231	E7	Á	Option-e, Shift-a	Á	Alt+0193
232	E8	Ë	Option-u, Shift-e	Ë	Alt+0203
233	E9	È	Option-`, Shift-e	È	Alt+0200
234	EA	Í	Shift-Option-s	Í	Alt+0205
235	EB	Î	Shift-Option-d	Î	Alt+0206
236	EC	Ï	Shift-Option-f	Ï	Alt+0207
237	ED	Ì	Option-`, Shift-i	Ì	Alt+0204
238	EE	Ó	Shift-Option-h	Ó	Alt+0211
239	EF	Ô	Shift-Option-j	Ô	Alt+0212

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
240	F0		Shift-Option-k	▨	▨
241	F1	Ò	Shift-Option-l (L)	Ò	Alt+0210
242	F2	Ú	Shift-Option-;	Ú	Alt+0218
243	F3	Û	Option-i, Shift-u	Û	Alt+0219
244	F4	Ù	Option-`, Shift-u	Ù	Alt+0217
245	F5	ı	Shift-Option-b	▨	▨
246	F6	ˆ	Shift-Option-i	▨	▨
247	F7	˜	Shift-Option-n	▨	▨
248	F8	¯	Shift-Option-,	¯	Alt+0175
249	F9	˘	Shift-Option-.	▨	▨
250	FA	˙	Option-h	▨	▨
251	FB	˚	Option-k	▨	▨
252	FC	¸	Shift-Option-z	¸	Alt+0184
253	FD	˝	Shift-Option-g	▨	▨
254	FE	˛	Shift-Option-x	▨	▨
255	FF	ˇ	Shift-Option-t	▨	▨

Note: 灰色で塗りつぶされたボックスは、Windowsで使用できない文字か、Macintoshと異なる文字を示しています。

ファンクションキーコード

4DはKeyCodeシステム変数にファンクションキーの値を返します。これはON EVENT CALL コマンドでインストールされたプロジェクトメソッド内部で使用されます。これらのプロジェクトメソッドはイベントをキャッチするために使用されます。

ファンクションキーの値はに基づいていません:

Function key	KeyCode
F1	-122
F2	-120
F3	-99
F4	-118
F5	-96
F6	-97
F7	-98
F8	-100
F9	-101
F10	-109
F11	-103
F12	-111
F13	-105
F14	-107
F15	-113

備考: KeyCodeシステム変数はON EVENT CALLでインストールされたプロジェクトメソッド内で使用されます。

ファンクションキーに加え、以下の表ではReturnやEnterを押した際にKeyCodeに返される値を示します。

Key	Code
Enter	3
Return	13
Backspace or Delete	8
Tab	9
Escape or Clear	27
Del	127
Help	5
Home	1
End	4
Page Up	11
Page Down	12
Left Arrow	28
Right Arrow	29
Up Arrow	30
Down Arrow	31

ランゲージリファレンス - 新着

- 14.0

On REST Authentication データベースメソッド New 14.0

ARRAY BLOB New 14.0

ARRAY OBJECT New 14.0

ARRAY TIME New 14.0

ARRAY TO LIST Updated 14.0

BLOB TO VARIABLE Updated 14.0

C_OBJECT New 14.0

COPY DOCUMENT Updated 14.0

Current form name New 14.0

DISPLAY NOTIFICATION Updated 14.0

Document to text New 14.0

FONT LIST Updated 14.0

Font nameDeprecated 14.0

Font numberDeprecated 14.0

Form event Updated 14.0

FORM GET NAMES Updated 14.0

FORM LOAD Updated 14.0

FORM UNLOAD New 14.0

Generate digest Updated 14.0

GET ACTIVITY SNAPSHOT New 14.0

Get last update log path New 14.0

GET MEMORY STATISTICS Updated 14.0

GET STYLE SHEET INFO New 14.0

GRAPH Updated 14.0

GRAPH SETTINGS Updated 14.0

HIDE TOOL BARDeprecated 14.0

HTTP Get Updated 14.0

HTTP Get certificates folder New 14.0

HTTP Request Updated 14.0

HTTP SET CERTIFICATES FOLDER New 14.0

INTEGRATE MIRROR LOG FILE New 14.0

JSON Parse New 14.0

JSON PARSE ARRAY New 14.0

JSON Stringify New 14.0

JSON Stringify array New 14.0

JSON TO SELECTION New 14.0

LIST OF STYLE SHEETS New 14.0

LIST TO ARRAY Updated 14.0

LISTBOX DUPLICATE COLUMN New 14.0

LISTBOX Get array New 14.0

LISTBOX GET OBJECTS New 14.0

LISTBOX Get row color New 14.0

LISTBOX Get row font style New 14.0

LISTBOX MOVE COLUMN New 14.0

LISTBOX SET ARRAY New 14.0

LISTBOX SET ROW COLOR New 14.0

LISTBOX SET ROW FONT STYLE New 14.0

METHOD GET CODE Updated 14.0

METHOD SET ATTRIBUTE Updated 14.0

METHOD SET CODE Updated 14.0

OB Copy New 14.0

OB Get New 14.0

OB GET ARRAY New 14.0

OB GET PROPERTY NAMES New 14.0

OB Get type New 14.0

OB Is defined New 14.0

OB Is empty New 14.0

OB REMOVE New 14.0

OB SET New 14.0

OB SET ARRAY New 14.0

OB SET NULL New 14.0

OBJECT Get action New 14.0

OBJECT Get border style New 14.0

OBJECT Get context menu New 14.0

OBJECT Get data source New 14.0

OBJECT GET EVENTS New 14.0

OBJECT Get indicator type New 14.0

OBJECT Get list name Updated 14.0

OBJECT Get list reference New 14.0

OBJECT GET MAXIMUM VALUE New 14.0

OBJECT GET MINIMUM VALUE New 14.0

OBJECT Get multiline New 14.0

OBJECT Get placeholder New 14.0

OBJECT GET PRINT VARIABLE FRAME New 14.0

OBJECT Get style sheet New 14.0

OBJECT Get text orientation New 14.0

OBJECT Get three states checkbox New 14.0

OBJECT Get type New 14.0

OBJECT Is styled text New 14.0

OBJECT SET ACTION New 14.0

OBJECT SET BORDER STYLE New 14.0

OBJECT SET CONTEXT MENU New 14.0

OBJECT SET COORDINATES New 14.0

OBJECT SET DATA SOURCE New 14.0

OBJECT SET EVENTS New 14.0

OBJECT SET FONT Updated 14.0

OBJECT SET INDICATOR TYPE New 14.0

OBJECT SET LIST BY NAME Updated 14.0

OBJECT SET LIST BY REFERENCE New 14.0

OBJECT SET MAXIMUM VALUE New 14.0

OBJECT SET MINIMUM VALUE New 14.0

OBJECT SET MULTILINE New 14.0

OBJECT SET PLACEHOLDER New 14.0

OBJECT SET PRINT VARIABLE FRAME New 14.0

OBJECT SET RGB COLORS Updated 14.0

OBJECT SET STYLE SHEET New 14.0

OBJECT SET TEXT ORIENTATION New 14.0

OBJECT SET THREE STATES CHECKBOX New 14.0

On Host Database Event データベースメソッド New 14.0

OPEN COLOR PICKER New 14.0

OPEN FONT PICKER New 14.0

PAUSE INDEXES New 14.0

QR ON COMMAND Updated 14.0

QR REPORT Updated 14.0

QR SET DESTINATION Updated 14.0

RESTART 4D New 14.0

RESUME INDEXES New 14.0

Selection to JSON New 14.0

SET DATABASE PARAMETER Updated 14.0

SET DRAG ICON New 14.0

SET RECENT FONTS New 14.0

SET UPDATE FOLDER New 14.0

SHOW TOOL BARDeprecated 14.0

SPELL CHECK TEXT Updated 14.0

SPELL GET DICTIONARY LIST Updated 14.0

SPELL SET CURRENT DICTIONARY Updated 14.0

ST COMPUTE EXPRESSIONS New 14.0

ST FREEZE EXPRESSIONS New 14.0

ST Get content type New 14.0

ST Get expression New 14.0

ST GET OPTIONS New 14.0

ST Get plain text Updated 14.0

ST GET URL New 14.0

ST INSERT EXPRESSION New 14.0

ST INSERT URL New 14.0

ST SET OPTIONS New 14.0

TEXT TO DOCUMENT New 14.0

Time Updated 14.0

Validate password Updated 14.0

VARIABLE TO BLOB Updated 14.0

Version type Updated 14.0

WA Evaluate JavaScript Updated 14.0

WA EXECUTE JAVASCRIPT FUNCTION Updated 14.0

WA SET PREFERENCE Updated 14.0

WEB SEND TEXT Updated 14.0

WEB SERVICE CALL Updated 14.0

ランゲージリファレンス - コマンドリスト (文字順)

-
- A
- B
- C
- D
- E
- F
- G
- H
- I
- J
- K
- L
- M
- N
- O
- P
- Q
- R
- S
- T
- U
- V
- W
- X
- Y
- _

On REST Authentication データベースメソッド New 14.0